

# SCT-DCS MONITORING SOFTWARE

S.G.Basiladze (MSU), R.Brenner (Uppsala)

## Contents

<b>1. Introduction</b>	<b>2</b>
1.1. Software design aims and main features	2
1.2. The main structure elements of DCS	3
1.3. The types of DCS information	6
<b>2. User tools for dealing with DCS</b>	<b>7</b>
2.1. Names organization of the Detector Tree	7
2.2. Data Base, Configuration files and their levels	8
2.3. Calibration algorithms	10
2.4. Watching/Community groups	11
2.5. Status byte convention	11
<b>3. Core software in “C”</b>	<b>12</b>
3.1. The hierarchy of DCS actions	12
3.1.1. The “far side” Functions and they numbers	13
3.1.2. Transactions, structure of messages	14
3.1.3. The “near side” Requests and Responses	16
3.2. Scan and Event waiting Loops	17
3.3. Control Functions and Script Tables/Files	19
3.4. Error handling	20
<b>4. LabView version of SCT DCS software</b>	<b>21</b>
<b>5. PVSS application</b>	<b>22</b>
5.1. Indicator & Control Panels and Objects	22
5.2. PVSS Data Points structure	27
5.2.1. Data Points loading from the Data Base	29
5.3. PVSS Calls of “C”-functions	29
<b>6. Calibration software</b>	<b>30</b>
<b>7. Testing and Simulation software</b>	<b>32</b>
7.1. Manual Controller in “C”	33
7.2. CAN-bus analyzer	35
<b>8. Supplements</b>	<b>37</b>

# Introduction

## 1.1. Software design aims and main features

Wide Range and Usage Simplicity. The first aim of the project was to make “scalable” software that allows to create a simple local test stations (1-2 FieldBus boxes with several tens of channels) and from other hand to build the complex geographical distributed slow control systems that may contain 5-6 hierarchical levels and thousands of the channels (FieldBuses are on the down level(s) in that systems).

All the manipulations with the nodes (and their branches) in hierarchical system structure (tree) are based on a simple set of functions such as Open/Close, Set/Get Data/Status, etc. This set of functions is hardware and protocol independent and may be used on the any hierarchical level.

All the important for user and changeable parameters of monitoring and control are placed into the configuration file(s) and may be changed without software recompiling. The original configuration files are creating, filling and editing as the tables in Microsoft Access Data Base.

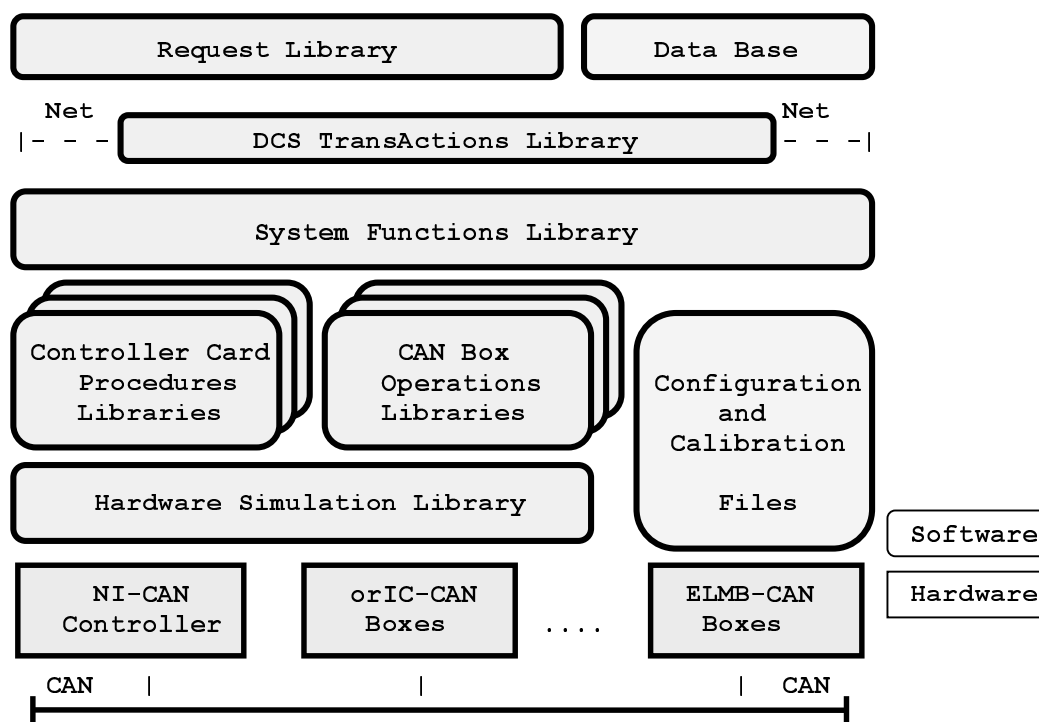
Created software is platform independent – any computer under Unix, Linux, Windows (and VMS – historically) may be used in the system (except user interface).

Hard-Open. Any type of FieldBuses and FieldBus nodes may be used in the system as well as some specific interconnection protocols (PC parallel and serial ports, GPIB buses, VME buses, etc.) in addition to the main CAN-Open protocol.

The current software is able to maintain 2 versions of CAN-controller cards, 8 versions of CAN-Boxes, Serial & Parallel ports of IBM PC and VME –bus modules.

Flexibility and High Performance. The flexibility is supported by hierarchical modular organization of the software; the main structure elements (libraries) are shown in Fig.1.

Figure 1



As it may be easy seen from Fig.1, there is some hierarchical set of actions in the system: operations, procedures, functions, transactions and requests. The operations are performing on the driver (CAN-box) level, the procedures – on Controller card (CAN-Open) level, the system

functions are hardware independent and on the user level the system requests are generating. The transaction level is just an intermediate layer between requests and functions.

The main software components are realized in “C”, that ensures the high execution speed and efficiency of actions. The desirable configuration of DCS system (operation system, set of libraries) may be selected by using “#ifdef - #else - #endif” keys before compilation. In particular, the Transactions library (Client/Server mode) may be included or excluded (simulated). Due to it the code has no redundancy practically and is very compact.

Powerful Diagnostics, Testing and Simulation. The “on-run” diagnostics means that:

- an ability of the node to perform requested action is checking before the command execution;
- the result of execution on the every step is checking, in the case of fault on some level the full hierarchical pointer to incorrect operation (address and function, starting from upper layers) is represented in an error code;
- if network connection is used than every “near side” request is confirmed from “far side” by feedback message.

The special testing programs were designed for checking of DCS hardware and two simulation mechanisms are built into the software:

- data simulation, the random values around Normal physical value may be generated in this mode;
- error simulation, the random errors with desirable probability are generated in that mode, it allows to test system stability and tolerance to possible deviations out of normal operation.

## 1.2. The main structure elements of DCS

Preliminary NOTE: all the terms that are defined in this section are started from Capital letter in the followed text (because they have a specified meaning in the boundaries of this document only).

The beginning notions of SCT DCS are the following: *User* (physicist) and *Expert* (system creator or maintain person); *Detector Tree*, *DCS Tree* and their *Agent*.

**Detector Tree** is the hierarchical structure of SCT-detector itself (see Fig.2 from the right). It may contain up to 6 levels, the main of them are the following:

- *Detector* level, includes SCT-detector as a part of ATLAS;
- *Section* level, maybe the barrel cylinders or the forward tracker disks;
- *Sector* level, the staves (parts of cylinder) or sectors (parts of disk);
- *Module* level, the microstrip detector modules of a stave or disk;
- *Sensor/Actuator* level, the module channels for monitoring or control.

**DCS Tree** - the hierarchical structure of Detector Control System (DCS) for SCT-detector (see Fig.2 from the left). It may contain up to 6 levels, the main of them are the following:

- *System* level, based on Local Control Station (LCS) – IBM PC and is a part of ATLAS DCS;
- *Host* level, the subparts of SCT DCS based on Satellite Control Stations (SCS) – IBM PC also;
- *Node* level, PC controller cards for FieldBuses or local communications;
- *Port* level, the ports of CAN-controller card, or PC serial/parallel port, or a port of Ethernet card;
- *Unit* level, the devices that are connected to CAN-bus, or to other serial or parallel links;
- *Channel* level, the parts of Unit that get a monitoring data from one Sensor or put control data to one Actuator.

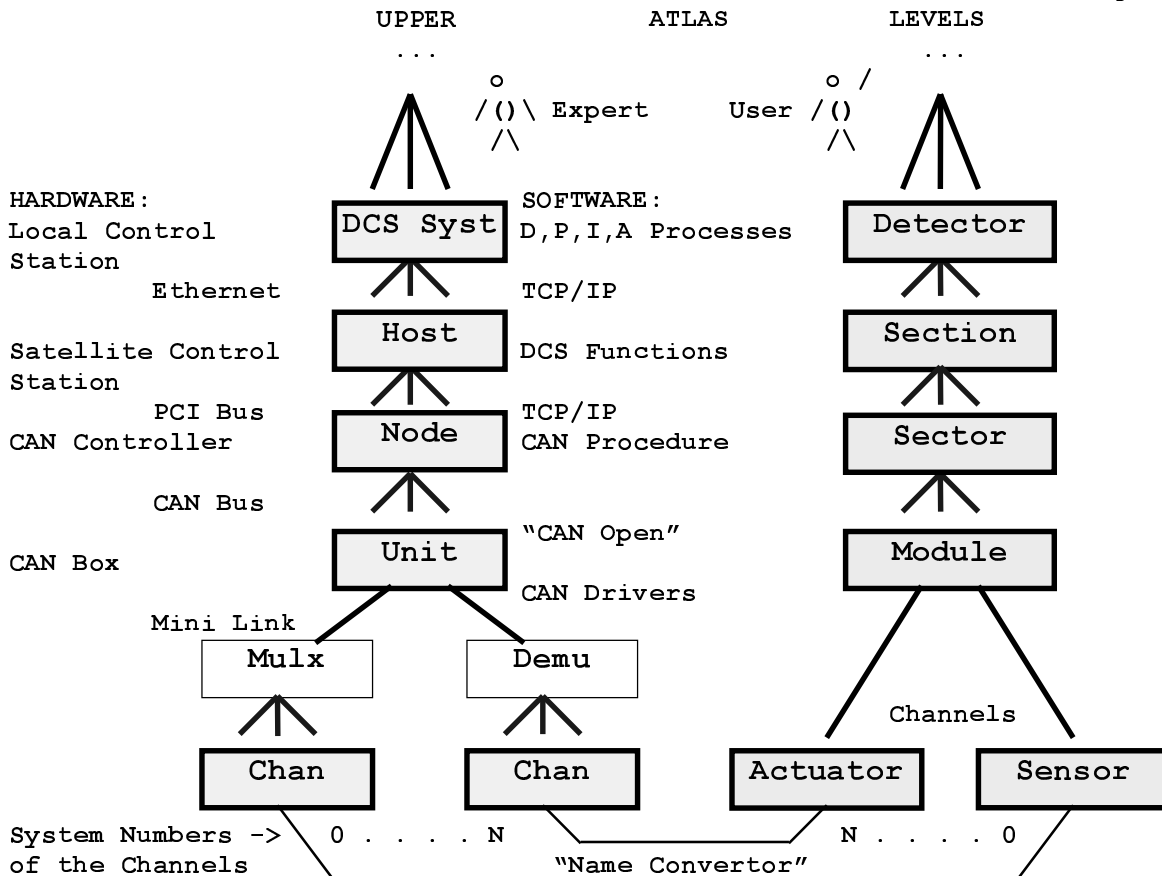
**Agent** – tree node on the any level that has brunches to the down Agents (if level > 1).

WARNINGS: 1. The term “Agent” is used here instead of usual term “node” (of tree) because “Node” is occupied for the 4-th level of DCS Tree.

2. The CAN-Open node (CAN box) is a Unit in this document.

The both structures have a common root and (on the down level) the equal number of channels, but the number of intermediate levels and the number of Agents on the every level may be different.

Figure 2



Agents Names and Addresses. As it is well known, every Agent identification may be represented in the following forms:

- *Name* , readable text string;
- *Address*, “numerical name” – set of numbers.

The Names are using here for the Detector Tree Agents, i.e. for sensors identification (they are oriented on a User) and the Addresses are using for the DCS Tree Agents, i.e. computers, cards, boxes and channels identification (they are necessary for Expert).

In principle, every Name may be individual, but nobody is able to remember a large set of Names, due to it the hierarchical (local) Names are using only because they reflect the structure of Detector Tree and the set of local Names is relatively small.

The Addresses are organized in a hierarchical manner also – local Addr(ess) is just a number of channel in a box, box in a port, etc. The delimiter between the local Iden(tificator)s is “.” for both - Names and Addresses.

Local and System Numbers. It is supposed that DCS Tree has “full” and fixed structure (logically): every Agent on some level has an equal number of subAgents (not all of them may be occupied physically). In this simplified case:

*Local Number* is a serial number of Agent in the group that belongs to one upper Agent;

*System Number* is a total serial number of Agent on its hierarchical level (see Fig.2). The Host, Node, Port, Unit, Chan(nel) terms are used in DCS Tree for its hierarchical Agents definition and for their Local Number definition as well:

Host - the serial number of Power PC (if many);  
Node - the serial number of CAN Controller card in one Host;  
Port - the serial number of CAN Controller Port (or CAN bus) in one Node;  
Unit - the serial number of CAN-box on the CAN-bus (for one Port);  
Chan - the serial number of CAN-box channel.

If the DCS Tree is full it may be described by limited set of numerical values – every value represents the maximum number of Agents on appropriate level: HOSTmax, NODEmax, PORTmax, UNITmax, CHANmax; than the System Number of any Agent in DCS Tree may be easy calculated:

for a Node       $Nnum = (Host * NODEmax) + Node;$   
for a Port       $Pnum = (Host * NODEmax * PORTmax) + (Node * PORTmax) + Port;$   
for a Unit       $Unum = (Host * NODEmax * PORTmax * UNITmax) + (Node * PORTmax * UNITmax) + (Port * UNITmax) + Unit.$

These system numbers are non-visible usually neither for User nor for Expert, but they are widely used in the software (as indexes) for access to Node and Unit structures.

In addition to the individual System Addresses two formal “to-all” and “to-nobody” codes (100 and 99 now) may be used for addressing. The tALL code permits to have an access to all the Agents of selected level and toNOB excludes any access on a current level:

Node System Iden (Addr): Host . Node . tNOB . tNOB . tNOB  
Unit System Iden (Addr): Host . Node . Port . Unit . tALL  
Chan System Iden (Addr): Host . Node . Port . Unit . Chan

For instance, the first line in the upper case means that access is performing to the Node, but not below; the second line means that access (with the same System Function) is performing to the all channels of selected Unit; the last line means the individual access to the Chan.

Multiplexing and demultiplexing in CAN. Every CAN message on a CAN bus usually has a Data Fragment that contains the set of bytes (say – 8). One or two bytes correspond to 1 Cell that is carrying Data from/to one DCS Chan. For example, the Data from 3 channels (16 bits) may be sent in one CAN frame (the 7-th byte may be used as a Chan Iden and the last one for a status).

Every CAN box may have several CAN frames with the individual internal addresses (for instance, PDO1 and PDO2 in CAN Open); it means that DCS Chan-s are organised inside CAN box as internal *two level* structure – Data Cells are combined in the CAN *Fragments* (frames). The local serial number of channel in DCS Address Tree may be calculated in a similar way:

$$Chan = (Frag * CELLmax) + Cell,$$

where CELLmax – maximum possible number of Cells in one Frag(ment).

Because the internal CAN box organisation may be very different (for instance, the ATLAS ELMB box has one CAN cell, but it is sending Data from 32 Chan-s in serial) the internal structure of MULTiplexors and DEMULTiplexors (i.e. frames and cells) is “transparent” for User (as it is shown in Fig.2) and for him one DCS Unit contains the DCS Chan-s only. The only place where frames and cells may be “visible” is the Chan *format* in the Unit Configuration file.

While sending the data into one DCS Chan only, some care should be taken about other Data Cells in the same CAN frame (they should be filled by appropriate Data for other DCS

Chan-s). The way was chosen for obtaining the “frame coherence” is to keep the copy of current data of CAN cells in the software structures that are associated with the Unit.

Over Nodes. Sometimes (for Control mainly) maybe necessary to use the Nodes with a minimal number of internal Units (say 1); the good example is parallel Port of PC. If the usual Node (and its branches) of DCS Tree will be used for such Agent than the large number of software structures for Units ((PORTmax \* UNITmax) - 1) will be empty but they will occupied the computer memory.

The special “Over usual Scan Tree” Nodes may be used in this case. It may have several Over Ports but every Port has one Unit only. The Over Nodes are not included in scanning procedure (see below) but they are available for individual requests.

### 1.3. The types of DCS information

**Data** – a portion of information from/to Sensor/Actuator; the Data may be

- *Raw* , means the Data will be sent to (were taken from) channel directly,
- *Physical*, Data in float representation that were converted from the Raw data by using specified calibration formula (it may contain up to 4 calibration constants).

**State** – every Sensor or Actuator channel and upper Agents have 32-bits Control Status Register (CSR), it contains an Agent *state* (On/Off) and *status* (Normal/Error). The upper 16 bits may be sent as CSR word to external device connected to a channel.

**Info** – the common term for Data and/or Stat(e) of a channel.

The Data should be represented for User in the Physical form (temperatures, voltages, currents, etc.) and conversion from Raw ADC counts to calibrated data is performed on the “far” side (where Data are taking) but not on the “near” side where User is requesting them.

Every DCS Agent and channel Data have a status that contains 4 conditions, the following colors are associated with them:

- |                                 |                                   |
|---------------------------------|-----------------------------------|
| - <i>Normal</i> status – Green, | - <i>Warning</i> status – Yellow, |
| - <i>Alarm</i> status – Red,    | - <i>Fatal</i> status – Cyan.     |

The state of the *every* Agent on *any level* has 4 conditions also:

- |                                                            |                                                       |
|------------------------------------------------------------|-------------------------------------------------------|
| - <i>Off</i> (the power may be Off, or no initialisation), | - <i>Idle</i> (Agent was initialised but not in use), |
| - <i>Ready</i> ,                                           | - <i>Active</i> .                                     |

Every Agent of slow control system needs usually some Configure information from the upper system layers for starting up. The channels should have in addition the Calibration information for converting Raw values to the unified (physical) Data for the upper layers. The upper layer Agent should be able to get (set) the current information about the State(s) of monitoring devices. The Synchro information may be necessary also for co-operative work of all the DSC parts.

In principle, the inter-Agents information exchange may be driven

- *by Call* (on Info Request from the upper level);
- *by Event* (on Info Sending from the down level);
- *by Loop* (periodically).

In the slow control system the first or the third mechanisms is using mainly, except some extra cases (for software Interlock) where the second type have to be used for obtaining the minimum *detection time* of a fault.

Every Request should have a Response in the *TimeOut* limits. The TimeOut value is a part of Config Info.

## 2. User tools for dealing with DCS

### 2.1. Names organization in the Detector Tree

The hierarchical Sensor Name reflects sensor position and/or function in the monitored part of Detector. The Local Name have to be short but quite understandable and its length should be constant preferably (the practice shows that 4 letters is near to optimum). Because the number of Agents with some (typical) Name may be large enough the additional *Index* for the name is necessary also. Below is a list of the Local Names that were used for Cooling sub-system.

Table 1

Sensor Names				
General:	Short:	Detailed:		Short:
Alignment	Algm			
Cooling	Cool			
Current	Curr			
		High (Volt)	Curr	HivoCurr
Humidity	Humi			
		Ambient	Humi	AmbiHumi
Interlock	Lock			
		Hardware	Lock	HardLock
		Software	Lock	SoftLock
Mechanics	Mech			
OverCurrent	Over			
		High (Volt)	Over	HivoOver
		Low (Volt)	Over	LovoOver
Pressure	Pres			
		Barometric	Pres	BaroPres
Temperature	Temp			
		Ambient	Temp	AmbiTemp
		Cable	Temp	CablTemp
		Cooling	Temp	CoolTemp
		High Voltage	Temp	HivoTemp
		Low Voltage	Temp	LovoTemp
		Module	Temp	ModlTemp
		Shield	Temp	ShldTemp
		Stave	Temp	StavTemp
Voltage	Volt			
		High	Volt	HighVolt
		Low	Volt	Low_Volt
Actuator Names		(will be added)		
Detector Names				
		Comments:		
Module level :		Module		
Sector level :		Stave		(Barrel )
		Sector		(Forward)
Section level :		Layer		(Barrel )
		Disk		(Forward)
Detector level:		SCT_Barrel		
		SCT_Forward		

For instance, the full name of detector module Sensor may be:

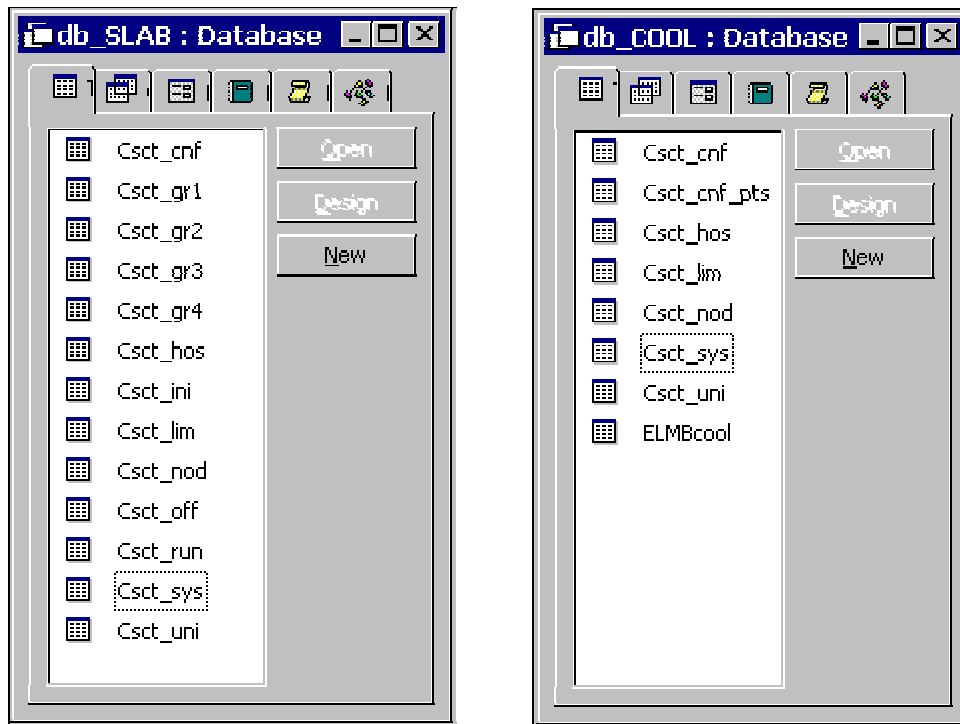
SCT\_Barrel . Layer04 . Stave03 . Module02 . HighVolt01

User usually deals with the Names only and an “C”-code converts them to DCS addresses “on the fly” (User is relatively slow “control device”, his reaction time is 300-400 ms, this time is enough even if the conversion subroutine is calling from interpreter).

## 2.2. Data Base, configuration files and their levels

The examples of Data Bases are shown in Fig.3, the left one is using for Testing (“C”) and for SiLab (LabView), the right one was done for SCT Cooling (PVSS).

Figure 3



The main Data Base tables are the following (see next page):

- *Csct\_sys*, SCT\_SYSem configuration, shows (but not defines) the DCS Tree constants and what History and Error files will be used in the Run;
- *Csct\_hos*, SCT\_HOSt configuration, defines Input/Output queues lengths, TimeOut values, the period of monitoring, the number of scanning loops in the Run and simulation mode;
- *Csct\_nod*, SCT\_NODE configuration, describes the physical (manufacture and production) names the logical Idens and initial Status of DCS Nodes and their Ports;
- *Csct\_uni*, SCT\_UNIt configuration, describes the same parameters for Units;
- *Csct\_cnf*, SCT\_CoNFig table shows the sensors Names, Chan-s addresses and their initial status, calibration constants, types of calibration formula and limit;
- *Csct\_lim*, SCT\_LIMits, gives 16 complex values for the *limit-types* (the typical values of Normal/Warning/Alarm/Fatal limits have a number as an Iden).


Every configuration and/or calibration Info may be changed by User before a Run and then it should be exported to the Bin-directory of a project as a plain text file. The Hosts must be placed in the Tables in the serial incremental order because every Table is reading only until the end of found Host section. The end of all the tables is marked by the “99” (unreal) number of a Host.

Figure 4

The System, Host, Node, Unit configuration Tables and the left columns of Chan-s Table

Microsoft Access

File Edit View Insert Format Records Tools Window Help



Cscst\_sys : Table

Index	HostMax	NodeMax	PortMax	UnitMax	ChanMax	LimiMax	OverNode	OverPort	IniFile	RunFile	OffFile	
1	1	1	1	1	8	64	16	2	2	100	100	100

Record: 1 of 1

Cscst\_hos : Table

Index	Host	Manufact	Product	Iden	SendMax	TakeMax	SendOut	TakeOut	EventOut	MoniPerd	ScanN
1	0 IBM	IBM_PC	PCEPHC103		100	200	0	10	50	20	
2	99 None	None	None		0	200	0	10	50	30	

Record: 1 of 2

Cscst\_nod : Table

Index	Host	Node	Manufact	Product	Iden	nSTS	p00Iden	pS00	Baud00	p01Iden	pS01	Baud01	p02Iden	pS02	Baud02	p03Iden	pS03	Baud03	Comment
1	0	0	NatInstr	NI-CAN	AtiCool	4	CAN0	4	250000	CAN1	0	250000	None	0	0	None	0	0	Scan Node
2	0	1	IBM_PC	PC_PP	AtiCool	4	LPT1	14	0	LPT2	0	0	None	0	0	None	0	0	Over Node
3	0	2	IBM_PC	PC_SP	AtiCool	4	COM1	C4	9600	COM2	C4	9600	None	0	0	None	0	0	Over Node
4	99	3	None	None	None	0	None	0	0	None	0	0	None	0	0	None	0	0	

Record: 1 of 4

Cscst\_uni : Table

ID	Host	Node	Port	Unit	Manufact	Product	CANId	uSTS	Leng	Frame	Time	SensRefer	Comment
1	0	0	0	0	ATLAS	ELMB	1	C4	2	8	1.2.496	Scan Unit	
2	0	0	0	1	ATLAS	ELMB	2	C4	2	8	1.2.496	Scan Unit	
3	0	0	0	2	ATLAS	ELMB	3	C4	2	8	1.2.496	Scan Unit	
4	0	0	0	3	ATLAS	ELMB	4	C4	2	8	1.2.491	Scan Unit	
5	0	0	0	4	ATLAS	ELMB	5	C4	2	8	1.2.498	Scan Unit	
6	0	0	0	5	ATLAS	ELMB	6	C4	2	8	1.2.492	Scan Unit	
7	0	0	0	6	ATLAS	ELMB	7	C4	2	8	1.2.493	Scan Unit	
8	0	1	0	0	IBM_PP	PP_DEV	0	14	1	8	0.0.0	Over Unit	
9	0	1	1	0	IBM_PP	PP_DEV	0	10	1	8	0.0.0	Over Unit	
10	0	2	0	0	IBM_SP	SP_DEV	0	54	1	8	0.0.0	Over Unit	
11	0	2	1	0	IBM_SP	SP_DEV	0	54	1	8	0.0.0	Over Unit	
12	99	U	U	99	None	None	U	U	1	8	U.U.U		

Record: 1 of 12

Cscst\_cnf : Table

Level05	Level04	Level03
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01
SCT	Layer01	Loop01

Record: 1 of 492


Datasheet View

Figure 5

The upper part of Chan-s configuration/calibration Table

Microsoft Access

File Edit View Insert Format Records Tools Window Help



Cscst\_cnf : Table

Level04	Level03	Level02	Level01	Eno	Host	Node	Port	Unit	Chan	Stat	Cal_A	Cal_B	Cal_C	Cal_D	Conv	Limit	Refer	Comments
Layer01	Loop01	Outlet02	DackPres01	!	0	0	0	0	0	1.44	0	1.0354	0	1000	40	2	99	Pressure
Layer01	Loop01	Stave03	InjePres01	!	0	0	0	0	0	2.44	0	1.0417	0	1000	40	2	99	Pressure
Layer01	Loop01	Stave04	InjePres01	!	0	0	0	0	0	0.04	0	1.0434	0	1000	40	2	99	Pressure
Layer01	Loop01	Stave03	CoolTemp01	!	0	0	0	0	1	8.44	2	1.5244	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave03	CoolTemp02	!	0	0	0	0	1	9.44	-6	1.5277	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave03	CoolTemp03	!	0	0	0	0	1	10.44	-9	1.5202	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave03	CoolTemp04	!	0	0	0	0	1	11.44	-5	1.5321	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave04	CoolTemp01	!	0	0	0	0	1	12.44	0	1.5153	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave04	CoolTemp02	!	0	0	0	0	1	13.44	6	1.5230	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave04	CoolTemp03	!	0	0	0	0	1	14.44	2	1.5172	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave04	CoolTemp04	!	0	0	0	0	1	15.44	-5	1.5193	124.18	7.9491	30	1	99	NTC
Layer01	Loop01	Stave03	StavTemp01	!	0	0	0	0	2	24.44	14	1.0137	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp02	!	0	0	0	0	2	25.44	2	1.0170	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp03	!	0	0	0	0	2	26.44	-11	1.0167	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp04	!	0	0	0	0	2	27.44	9	1.0202	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp05	!	U	U	U	U	2	28.44	12	1.0178	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp06	!	0	0	0	0	2	29.44	-4	1.0105	257.44	3.0095	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp07	!	0	0	0	0	2	30.44	-12	1.0136	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp08	!	0	0	0	0	2	31.44	2	1.0190	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp09	!	0	0	0	0	2	0.04	-5	1.0212	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp10	!	0	0	0	0	2	1.44	4	1.0168	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp11	!	0	0	0	0	2	2.44	8	1.0164	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave03	StavTemp12	!	0	0	0	0	2	3.44	2	1.0175	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave04	StavTemp01	!	0	0	0	0	2	4.44	-7	1.0225	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave04	StavTemp02	!	0	0	0	0	2	5.44	3	1.0187	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave04	StavTemp03	!	0	0	0	0	2	6.44	17	1.0177	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave04	StavTemp04	!	0	0	0	0	2	7.44	14	1.0205	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave04	StavTemp05	!	0	0	0	0	2	8.44	2	1.0195	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave04	StavTemp06	!	0	0	0	0	2	9.44	-2	1.0188	257.44	3.8895	20	1	99	PT1000
Layer01	Loop01	Stave04	StavTemp07	!	0	0	0	0	2	10.44	9	1.0209	257.44	3.8895	20	1	99	PT1000

Record: 1 of 452

Datasheet View

### 2.3. Calibration Algorithms

The calibration part of “Csct\_cnf” Table contains the information for conversion of Raw values to Physical ones. The calibration procedure consists of 2 steps; on the first step the normalization of ADCs parameters (offset/pedestal and slope/gain) is performed, i.e. the electronics itself is calibrated. On the second step the sensors parameters are normalized to typical values. As was said before the number of calibration constants is equal to 4, the current calibration formula for the first step is the following:

$$s = B*(x - A),$$

where  $s$  - is the sensor value from “ideal” ADC,

$x$  - is the real ADC count (integer),

$A, B$  - are the first step calibration constants.

The following base approximations may be used on the second step (the number of calibration formula, for using in “Csct\_cnf” Table, is shown from the right):

linear approximation

$$y = (s - C)/D, \quad \text{if } s = D*y + C \quad (1)$$

square root approximation

$$y = [\text{SquareRoot}(C*C + 4D*s) - C]/2D, \quad \text{if } s = D*y*y + C*y \quad (2)$$

logarithm approximations

$$y = C + [\ln(s)/D], \quad \text{if } s = \exp[D*(y - C)] \quad (3)$$

$$y = 1 / \{ [1/C] + [\ln(s)/D] \}, \quad \text{if } s = \exp\{D*[(1/y)-(1/C)]\} \quad (4)$$

where  $y$  - is the physical value (float),

$s$  - is the sensor value,

$C, D$  - are the second step calibration constants.

If  $A=0, B=1, C=0, D=1$  the Raw values will be on the “output” ( $y = x$ ) for calibration formula (1).

The complex sensors, for instance for humidity monitoring, may be described in calibration table also. The formula for calculating of the humidity is

$$h = H1/(H1 + H2). \quad (10,a)$$

Relative Humidity, RH is calibrated by:

$$RH = B*h - A. \quad (10,b)$$

The  $A, B$ -calibration constants should be placed in the record for the second sensor ( $H2$ ) and in its last column (Refer) should be presented the reference on the first sensor ( $H1$ ) local number in the same Unit.

Basing on upper equations the following calibration formula is using for the PT1000s on the second step:

$$T = (R / D) - C; \quad (20)$$

and for the NTC-sensors:

$$T = (1 / ((1 / C) + (\lg(R) / D) + (\lg^3(R) / 4700000))) - 273.15; \quad (30)$$

and for Pressure sensors:

$$P = (B * (x - A)) * 100 * (100 / 65536) / D; \quad (40)$$

(Divider->100 100<-Voltage Range, mV)

where  $R$  - is the sensor resistance:  $R = s * Ra / (65536 * ((E - U) / 100));$

$Ra$  - is Adapter resistance (Ohm),

$E$  - is the ELMB-box reference voltage (mV),

$U$  - is the sensor voltage:  $U = s * 100 \text{ (mV)} / 65536;$

There are 2 additional conversion formulas for testing CAN-boxes. The first one represents Data in a Raw format (ADC counts):

$$y = x, \quad (*1)$$

the A,B,C,D-constants are ignored in this case; and the second one converts ADC count into the input (Sensor) voltage:

$$y = (s / 65536) * 100 \text{ mV}; \quad (*2)$$

where \* - may be 1, 2, 3 or 4 here (i.e. any conversion may be replaced by \*1 or \*2).

## 2.4. Watching/Community Groups

The Data from monitored sensors may be selected by using the various criteria's. Every criteria combines the proper channels into a

*Group* – the set of monitored Chan-s that are selected for sampling investigation of Data. For instance, the simplest way is to combine all the sensors of one type into one Group. The second way – to unite them basing on their “geographical area” community (for instance, for Ladder or Disk), etc.

The 8 selection Groups are possible in the current “C” (testing) software. The 8-bits array is using in the CSR (the second byte) of any channel as a (linear) Group code; it means that every Chan may belong up to the 8 Groups simultaneously. The testing software has no “screen object” for selecting and combining desirable channels into a Group, due to it the information about the Chan-s Addresses and their serial order in a Group is stored in “Cset\_gr1-8” files (see left Data Base in Fig.3).

NOTE: 1. The start Group number is ‘1’;

2. The start Order number is ‘1’ (‘0’ means the “channel absence” – may be used as a mask);

3. In the Request attributes the Group number corresponds formally to Chan number and the Order number corresponds formally to Input-Info (see 3.3).

The Chan-s that are listed in the Group-files are included automatically in the set of monitored channels.

The 16 selection Groups are available in PVSS environment for watching Data in graphical form. The pop-up panel may be called from an every Plot for (up to) 16 Data manual selection.

## 2.5. Status Byte convention

Lets repeat that every DCS Agent (on the Node, Port, Unit and Chan levels) has an unified “Control-Status Register” (CSR):

CAN Specific CSR	Group Code (bit array)								R	W		S	E		
								80	40	20	10	8	4	2	1

The meaning of R(ead), W(rite), S(tatus), E(rror) fields is shown below:

			HexCode:
E	Bit_0,1 - Severity of Error:	0 - Normal (No Error),	0h
		2 - Warning,	2
		1 - Alarm,	1
		3 - Fatal;	3
S	Bit_2,3 - Operation mode	: 0 - Module is OFF System,	0
		1 - Module is IN System,	4
		2 - Module is READY,	8
		3 - Module is ACTIVE (Busy);	C
		R: W:	
W	Bit_4,5 - Write Abilities	: 0 - No Ability,	00 00
R	Bit_6,7 - Read Abilities	: 1 - By Call (from UpplLevel),	40 10
		2 - By Event(from LowLevel),	80 20
		3 - Periodically.	C0 30

The “general” status information (2 low bytes) is writing to the every Agent CSR from the Config Files during the initialisation. The “W” and “R” fields describe the logic of Agent Info exchange (4 conditions - see 1.3), i.e. Agent abilities for writing and reading.

In addition to 2 general status bytes there are 2 CAN specific CSR bytes that may be used in the different ways. They may be written and/or read to/from external device by using the special DCS “send/take status” functions.

### 3. Core software in “C”

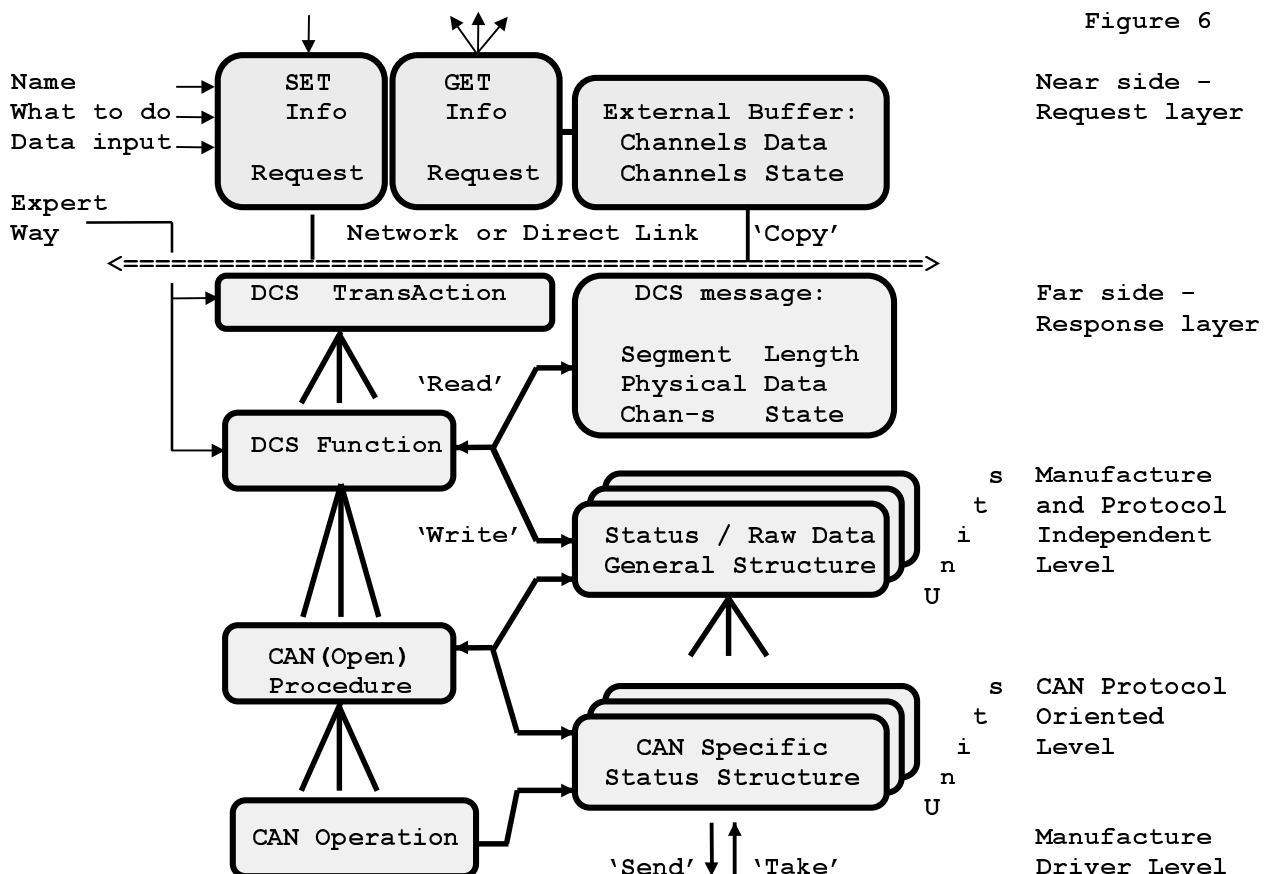
#### 3.1. The hierarchy of DCS actions

The CAN equipment is only the part (down level part) of the entire slow control system and CAN protocols can't cover all the needs of a complex system. For instance:

- “CAN Open” protocol is defined in the boundaries of one CAN bus only,
- the possible number of CAN cells for one COB-Id is very limited.

But the standard “CAN Open” protocols may be successfully included on the down level in the set of DCS protocols.

The current hierarchical organization of the main DCS actions is shown in the Fig.6. The based actions are “Write” and “Read” Data or Status; the additional: “Init” and “Close”.



The manufacture drivers and/or "CAN Open" profiles are the base (first level) elements of DCS actions hierarchy. The several simple CAN-box Operations are combined in one CAN Procedure for changing of CAN modules State or setting/getting the Data. The specific (vendor defined) Status structures are using on this level.

Every Node/Unit on the next level is represented by its Node/Unit *General Structure* which contains the main changeable Node/Unit attributes (parameters) and Raw Data (for Unit). These Structures are initialized from the Config Files when an Init Function of the proper level is called. System Functions are using for Info exchange with General Structures on this level.

The next is so-called TransAction level where the CAN (or other) Info is placed into the *Message* buffer. There are 2 types of the DCS messages:

- *short* ones, where only one channel Data or Status is sending as an integer (raw) value,
- *long* ones, where the physical Data and Status information from all the channels is sending.

In the last case the monitored Data are converted in the calibrated form and the message information is "compactized" (a form of compression, see Sect.3.1.4). All the Data and Status *Requests* are the short messages. The *Response* DCS message maybe short or long. The format of DCS message is described below, the message buffer on the "far" side is storing the Info about one Scan only and it is rewriting periodically in the asynchronous mode.

The last - "Request" level permits to make setting and getting DCS Info by using the Names of Detector parts. Every Request may be a broadcast or broadcast command for CAN hardware because a Detector Name ("..."tALL") may correspond to a group of the CAN channels.

There are two possibilities in the interconnections between Request and Response layers (Fig.6):

- *remote mode*, real TCP/IP Client/Server connection from near to far side,
  - *local mode*, direct "near" to "far" internal connection (Net simulation);
- in the last case a DCS message is using just as internal Buffer for one Scan.

### 3.1.1. The "far side" Functions and they numbers

Below is a list of the main DCS Functions:

Table 2

0x10:	int	InitDCShost(	int	Host);
0x14:	int	InitDCSnode(	int	Host, int Node, int Port);
0x18:	int	InitDCSunit(	int	Host, int Node, int Port, int Unit);
0x20:	int	WriteDCSstatus(	int	Host, int Node, int Por#, int Uni#, int Cha#, ui Wsts);
0x24:	int	SendDCSstatus(	int	Host, int Node, int Port, int Unit, int Chan, ui Wsts);
0x28:	int	TakeDCSstatus(	int	Host, int Node, int Port, int Unit, int Chan, ui *Rsts);
0x2C:	int	ReadDCSstatus(	int	Host, int Node, int Por#, int Uni#, int Cha#, ui *Rsts);
0x30:	int	WriteDCSdata(	int	Host, int Node, int Port, int Unit, int Chan, ui Wdat);
0x34:	int	SendDCSdata(	int	Host, int Node, int Port, int Unit, int Chan, ui Wdat);
0x38:	int	TakeDCSdata(	int	Host, int Node, int Port, int Unit, int Chan, ui *Rdat);
0x3C:	int	ReadDCSdata(	int	Host, int Node, int Port, int Unit, int Chan, ui *Rdat);
0x48:	int	WaitDCSevent(	int	Host, int Node, int Port, int Unit, int Chan, ui *Edat, ui *Ests);
0x4C:	int	TakeDCSevent(	uc	*Host, uc *Node, uc *Port, uc *Unit, uc *Chan, uc *More, ui *Cdat, ui *Csts);
0x50:	int	CloseDCSunit(	int	Host, int Node, int Port, int Unit);
0x54:	int	CloseDCSnode(	int	Host, int Node);
0x58:	int	CloseDCShost(	int	Host);
uc -> unsigned char, ui -> unsigned int				
# => may be absent, if Port,Unit,Chan-number == tNOB -> Access To NoBody				

The “Write” and “Read” functions are writing and reading Data or full Status word to/from Unit General Structures; the ‘Send’ and ‘Take’ functions provide in addition the Data and CAN specific CSR exchange between Unit’s structures and CAN hardware if an appropriate DCS Chan is writable or readable (see 2.5).

The Functions are manufacture and CAN protocol independent and may be widely used for many types of Buses and Serial Lines. All the Functions return a "zero" or an Error Code.

### 3.1.2. Transactions, structure of messages

The DCS TransAction (see Fig.6) is an equivalent of the DCS Function but it is organized as internal DCS message with the fixed “byte ordered” platform independent format.

The request (*command* - from the left) and response (*execute* - from the right) messages have the usual fields:

COMmand	Message (to CAN)	EXEcute Message ( from CAN )
B00	: Message Number	Message Number+1 (exe_msg_num: even)
B01-02:	Message Length	Message Length (in Bytes )
B03	: To.. - Chan Iden	From - Chan Iden (or VME address )
B04	: To.. - Unit Iden	From - Unit Iden ( : )
B05	: To.. - Port Iden	From - Port Iden ( : )
B06	: To.. - Node Iden	From - Node Iden ( : )
B07	: To.. - Host Iden	From - Host Iden
B08	: What - Func Iden	What - Func Iden (CAN/VME Function )
B09-13:	From - Five Bytes	To.. - Five Bytes
B14-17:	Info - Reserve= 0	Info - Error Code (LowByte is in B14)
B18-21:	Info - Reserve= 0	Info - Error Addr (LowByte is in B18)
B22-25:	Date - Day, H,M,S	Date - Day, H,M,S (Day is in B22)
B26...	Data - of 1 Frame	Data - of 1 Frame (and Status info )

Every Message starts from a current (incremented) non zero Number, it is odd for COM\_message and even for EXE\_message. Two next bytes contain the message Length identifier. Five Bytes of an address (To..) are carried in the Bytes\_03-07. Function Code is placed in "What to do" Byte (B08) of a Message.

Client is starting by issuing of a Request Message (with the next odd number) for Data/Status reading or writing (the request attributes are placed in the Bytes\_26-29).

Server can control of CAN modules and can execute commands from the Client Requests. Server acknowledges every Message receiving (by incrementing B00 and repeating of B03-13 in the Reply Message) and if there was short Info request - includes Read Data and/or Status starting from Byte\_B26. In the case of some error detecting the Server puts the first byte of an error code in Byte\_B14 and the second byte - in Byte\_B15, in the next 2 bytes the Error Address code will be written.

Every Message has 4 bytes for Day(month)/Hour/Minute/Second Time stamp – in the Bytes\_B22-25.

The Info Coding in DCS Message for monitored Data. There is the hierarchical set of embedded segments for Host, Node, Port, Unit and Chan portions of Info in the CAN Message. The number of segments on the every level is equal to HOSTmax, NODEmax, PORTmax, UNITmax, CHANmax correspondingly. The every fragment is starting from its length (2 Bytes), due to it the CAN Message may be easy decoded and encoded.

The Chan Info contains Data and Status fields. They both are “half Byte” encoded: code starts from a current value length (the first half of Byte) and then the value itself (float or integer) is represented in binary-decimal code. For instance, for “zero value” coding the only one Byte is needed for 32-bit Integer (code = 01h) and 2 bytes are used (code = 0B 03h) for the Float.

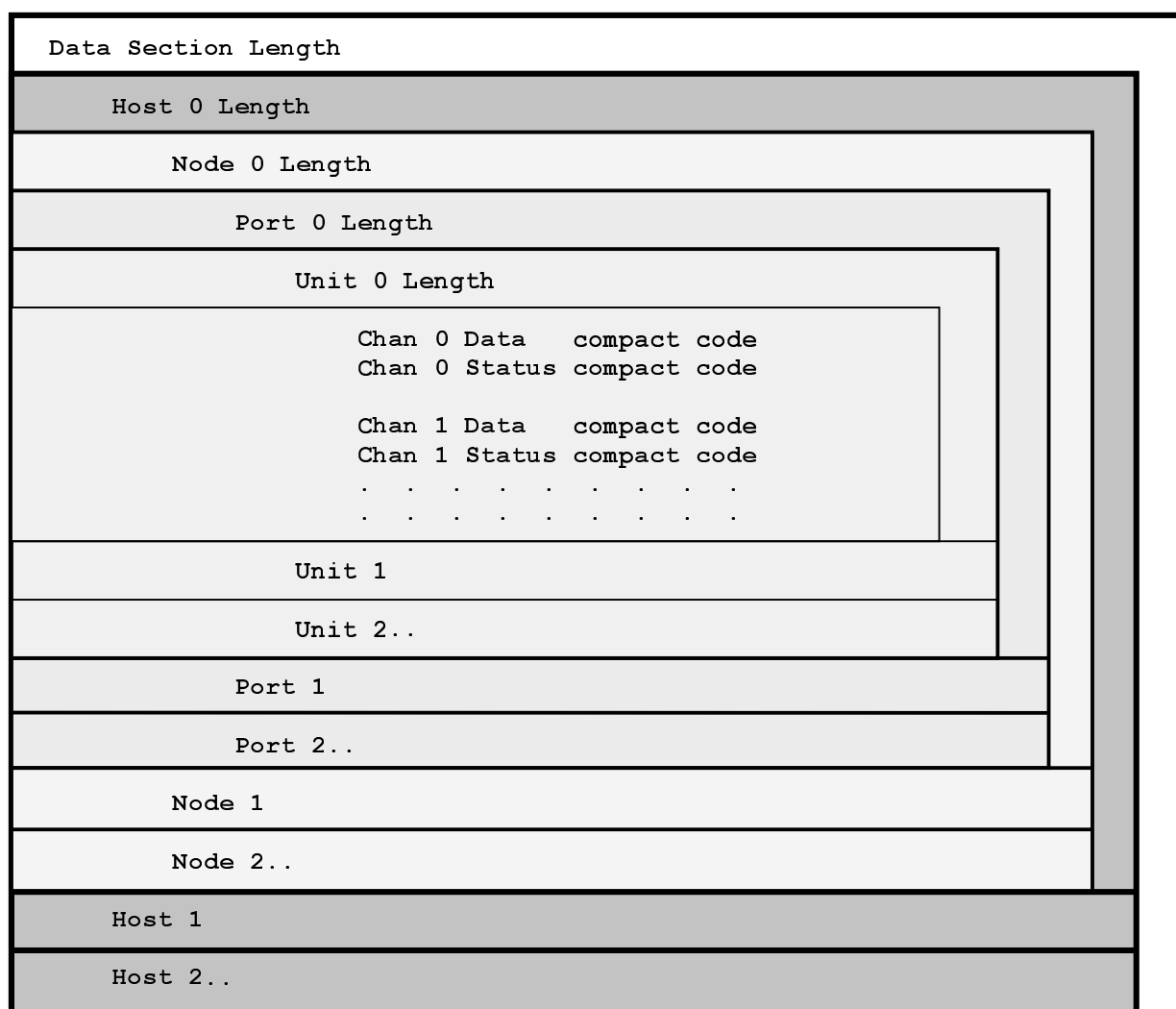
HalfByte (hB) codes for Numericals:

- 0..9 - Digits (Decimal),
  - 10 - Minus Sign,
  - 11 - Float Dot;
- Chars coding, if the First hByte:
- 12 - Char String, size = Length,
  - 13 - Char String, size = Length + 15,
  - 14 - Char String, size = Length + 30,
  - 15 - Char String, size = Length + 45.

Byte	
First hByte	Length in hB/B
Third hByte	Second hByte
. . . . .	. . . . .
(NULL)	Last hByte

The Name of Detector structure part may be coded also as a Byte Array. The length of string (up to 60 Bytes) is in the 'Length' hByte. The schematic structure of DCS-message is shown below:

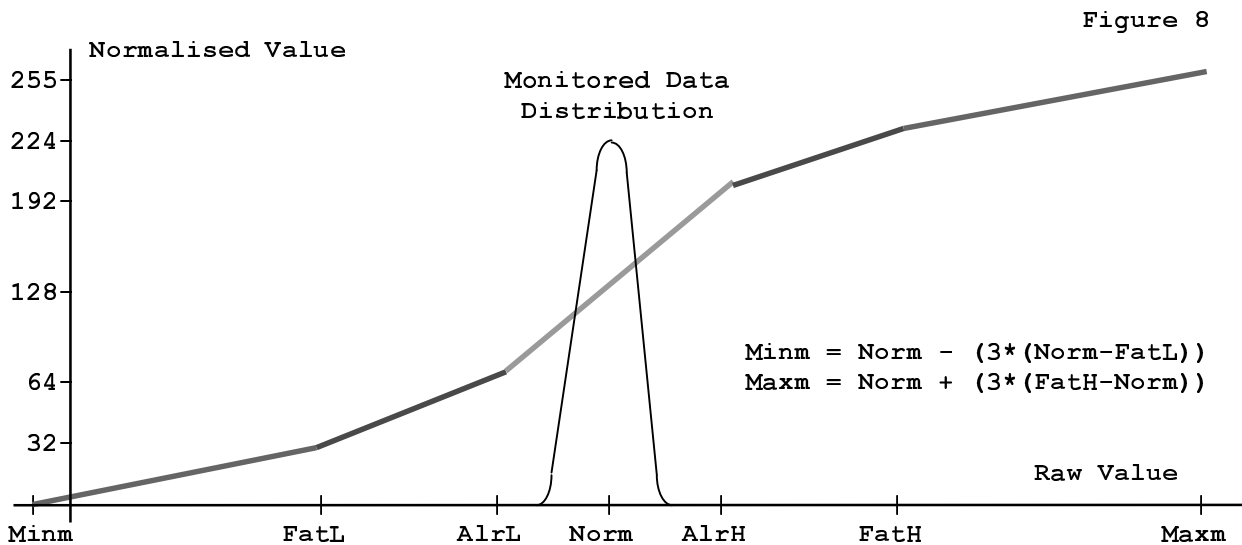
Figure 7



Data Normalization. As we need to know the detailed behavior of monitoring parameter around its Normal value only the "three slope" scale may be used as it is shown in Fig.8. For so-called Data Normalization procedure the Limits are standardized:

Alarm_Low is always equal to 64,	Alarm_High is always equal to 192,
Fatal_Low is always equal to 32,	Fatal_High is always equal to 224,
Minimum is always equal to 0,	Maximum is always equal to 255,

and Normal value is always equal to 128 - see Fig.8. As a result the 8 bits (1 Byte) only is necessary for monitoring Data (internal) representation. Such format is very suitable for graphics Data out and for the further Data Compression.



If  $FatL=n*32$ ,  $AlrL=n*64$ ,  $Norm=n*128$ ,  $AlrH=n*192$ ,  $FatH=n*224$  ("n" is any integer) are installed for the X-axe then the conversion scale will be linear (see LimiType "14" (n=1) and "15" (n=10) in the Calibration Table).

#### 3.1.4. The “near side” Requests and Responses

There are 2 near side high level subroutines for manipulating with the *single* channel Data and/or Status info (short messages). The first string in the attributes describes DCS Address of the channel. All the general Functions that are listed in p.3.1.1 may be used in the Request(). The CopyResponse() subroutine just takes the reading Info from the EXE\_message and may be used for checking the quality of Request execution.

```
int Request(int Host, int Node, int Port, int Unit, int Chan,
            int Func,                /* The DCS FUNCTION */
            unsigned int Info);      /* The Data/Status Info for an INPut */

int CopyRespond(int Host, int Node, int Port, int Unit, int Chan,
                unsigned int *Info); /* Pointer for Data or Status OUTput */
```

Two additional system functions may be requested also (see details in p.7):

```
0x80: int SetDCSstate(int Scod); - Set DCS System State
0x84: ui GetDCSstate(void);      - Get DCS Global state parameters
```

There are special Requests for Info in calibrated and “normalized” forms:

```
0x90: GetDCSinfo                - Function Code for Request
0x94: GetDCSnormal              - Function Code for Request (see p.5.2) ?
```

The full Data & Stat information from *all* the channels of the *last* Scan is sending in the compact format on this request. After it the small (Group) portions of Info may be *copied* from EXE-message locally.

The External Calls. For using “by call” from higher level system software the described set of core programs and subroutines may be represented as a single “functional box” with some inputs and outputs. In this case all the requests may be sent and all the responds may be taken (or copied from exe-message – see Fig.6) by using one complex higher level subroutine.

Two *External Calls* may be used for sending of the local or remote Requests to CAN hardware and receiving the Responses with Data/Status Info:

```
int SetGetByAddress(int *Host, int *Node, int *Port, int *Unit, int *Chan,
                  int *Func,      /* The DCS FUNCTION */
                  int *Iinp,      /* The Data/Status Info for an INPUT */
                  float *Dout,    /* Pointer for Data OUTPUT */
                  int *Sout,      /* Pointer for Status OUTPUT */
                  int *Eout)      /* Pointer for Error OUTPUT */

int SetGetByName( char *Name, int *Func, int *Iinp,
                 float *Dout, int *Sout, int *Eout,
                 int *Numb)      /* The NUMBER of Chans found by Name */
```

The first subroutine is oriented on Expert Requests because it deals with DCS addresses and the second one – on User Requests (Detector Name is in the arguments). The first subroutine returns (on Dout and Sout addresses) one Data and one Status value; the second one – the group of Data and Status values that was found “under” *broadcast* Detector Name - it may be the CAN-box, subComponent or all the Detector.

The writing and reading (mainly discussed before) are the examples of “single” action. For easy control of the system resources more complex control is necessary that is including a serial sequence of actions, such as scanning, opening and closing.

There are 2 *system requests* that may be called via “SetGet” subroutines for setting and getting the general DCS Status and for making the monitoring Scan:

```
0x80: int SetDCSstate(int Scod);
      where Scod - System Status CODE: 0x00 - Set OFF System State,
                                       0x04 - Set IDLE System State,
                                       0x08 - Set READY System State,
                                       0x4C - Set ACTIVE State and Scan Data,
                                       0xCC - Scan Data Periodically;
0x84: int GetDCSstate(void); - returns of Scod.
```

Two additional System Requests are defined for more specialized operations by using the Detector's Names:

```
0x88: SetDCSdata - for Data writing to the group of CAN channels
              (makes WriteDCSdata & SendCANdata);
0x90: GetDCSinfo - for copying the results to Data and Status Buffers.
```

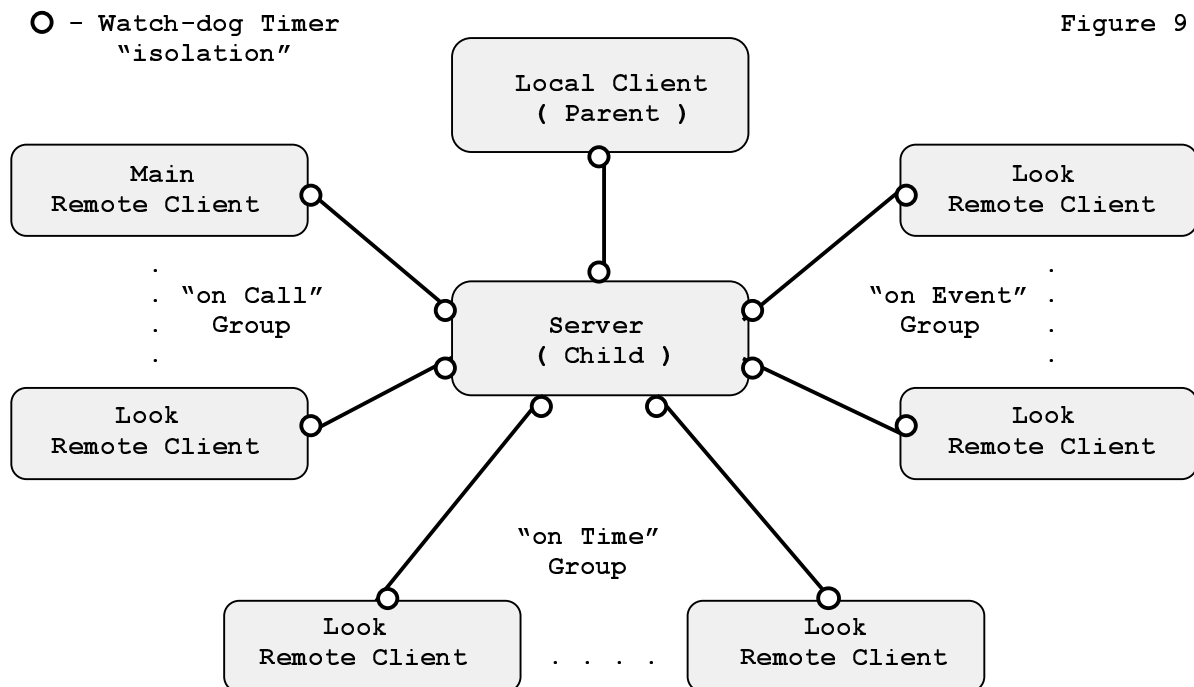
By using System Requests in the External Calls any extern connection to the DCS System (via interprocess mechanism or directly) may be done very easy. The core software, in particular, may be represented as DLL library and be used in any environment (LabView or PVSS).

### 3.2. Scan and Event waiting Loops

There are 2 types of the Clients in SCT-DCS (if TransAction library is included):

- "Main Client" that may be run by Expert only. It is able to generate any request to the Server;
- "Look Client" may be used by everybody. It is able to send "GetDCSinfo" request only.

The Server is starting by the Local Client as a “Child”, they both are sitting in a Host computer. The “Parent” is a Main client for the server. The Server executable program should be prepared before its call and should be put into the file with “Cserver.exe” name.



The other clients (their number may be up to 30) are subdivided in 3 groups: they can get slow control data “on Call” (by sending request every time they need) or “on Event” (that is discovered in the system) or Periodically (“on Time” – regular). The “event” and regular clients should register themselves before data getting by sending to Server a Request with GetDCSinfo function and with the Info value

- 0x88 for adding to Event list (0x84 – for deleting), see CSR bits in 2.5,
- 0xC8 for adding to Periodical list (0xC4 – for deleting).

After it they will receive the proper data from the Server.

The TCP/IP links are using for remote clients. All the links are realised in *unblocking* manner (there is no indefinite waiting time during “read” and “write” transActions ( see watch-dog timers in Fig.9), therefore nobody can cause the system hang-up.

The Server Loops (time sharing) organisation is shown in Fig.10. The average Detecting Time:

- for Scannable modules - tens or hundreds of sec (  $T_{mon} / \sqrt{12}$  );
- for Eventable modules - hundreds of msec (if Server is out of Scan).

The advantages of "Periodical Sending" mode in CAN (  $T_{send} \leq (T_{mon} / 2)$  ):

- CAN bus loading is more regular;
- there is no waiting time for response, therefore a Scan time is very short;
- there is no CAN-boxes TimeOut and hang-up problems.

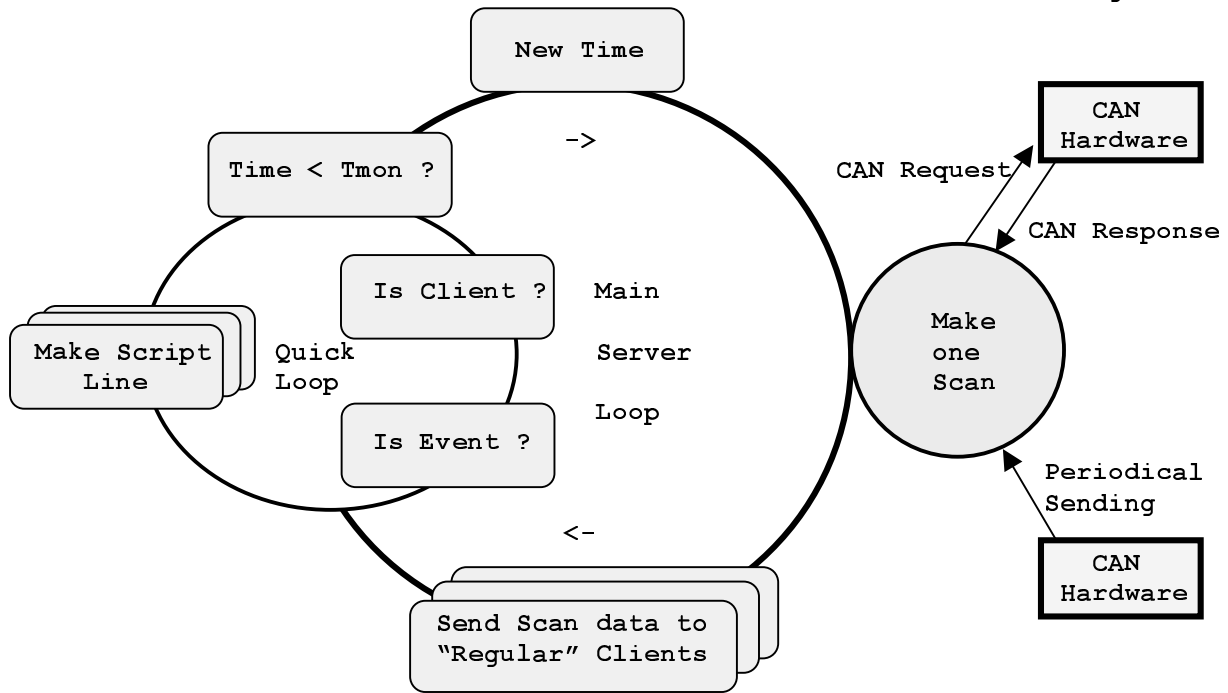
The CAN Unit is sending Data to Server only if the monitoring procedure was done successfully, in other case it returns an error.

Quick Loop procedures. In the 'Quick Loop' a Server is checking;

- the possible requests from Main and Look Clients;
- the 'Event List' that contains "look at me" Flags from the channels that are able to send the CAN messages in some Extra Cases;

until the Period of Monitoring (Tmon) is not over, after it Server starts new Main Loop again.

Figure 10



### 3.3. Control Functions and Script Tables/Files

The manipulation with the Actuator channels needs sending some control Info in non-predictable order (at the time of Code compiling) in addition to getting Actuators Data and Status from some addresses that may be varied also. The Info exchange protocol may be based on some other functions than simple numerical Value Write/Read (see p.3.1.1). For instance, it may be based on Text String Sending/Taking (cooling “Chiller” machine) and maybe very slow (needs several seconds waiting after every Request). The special system subroutine was created to meet these requirements, it is able to read and to execute the

*Script* - serial set of commands that are represented in the form of a text lines.

There are 3 Scripts in the core software now (see left Data Base in Fig.3):

- *Csct\_ini*, the script that is executing during system initialization,
- *Csct\_off*, the script that is executing during system closing,
- *Csct\_run*, the script that is executing “step by step” in a Quick Loop (see Fig.10).

One Script Line is executing in one Quick Loop cycle, the next Line is reading on the next cycle only. The important new function in a Script is ‘WaitMsec’ (milliseconds). If ‘WaitMsec’ is presented in the Line than incrementing of Sript Lines will be postponed till wait interval will be over (the Script itself will be transparent for Quick Loops).

The Functions that may be used in the script Requests and their “text format” (in the left columns) are shown below:

InitHost: func = INITdcsHOST	InitNode: func = INITdcsNODE
InitUnit: func = INITdcsUNIT	SendStat: func = SENDdcsSTATUS
ReadStat: func = READdcsSTATUS	TakeStat: func = TAKEdcsSTATUS
SendData: func = SENDdcsDATA	ReadData: func = READdcsDATA
TakeData: func = TAKEdcsDATA	SendStri: func = SENDdcsSTRING
TakeStri: func = TAKEdcsSTRING	MoveStri: func = MOVEdcsSTRING
ClosUnit: func = CLOSEdcsUNIT	ClosNode: func = CLOSEdcsNODE
ClosHost: func = CLOSEdcsHOST	WaitMsec: No Request on it

The format of the Script Line is the following:

	Request	Host	Node	Port	Unit	Chan	Func	Info/Stri
or	WaitMsec	Time	None	None	None	None	None	None

If some of the parameters are not necessary for the request (for instance, Node, Port, Unit, Chan for 'InitHost') they should be described as 'None'. Any String may be commented by inserting '#' before 'Request' or 'WaitMsec'.

Three additional functions were created for dealing with the *Strings*:

```

0x40: int  SendDCSstring( int Host, int Node, int Port, int Unit, int Chan,
                                char *Stri);
0x44: int  TakeDCSstring( int Host, int Node, int Port, int Unit, int Chan,
                                char *Stri);
0x9C:      MoveDCSstring  - FunctCode for Far side Request.

```

Every letter of a String[i] is copied (while Sending/Taking) into appropriate Chan[i] of the selected Unit structure. Taking is copied a String into sGET global buffer also. Due to it any String was read may be moved to any Unit.

### 3.4. Error Handling

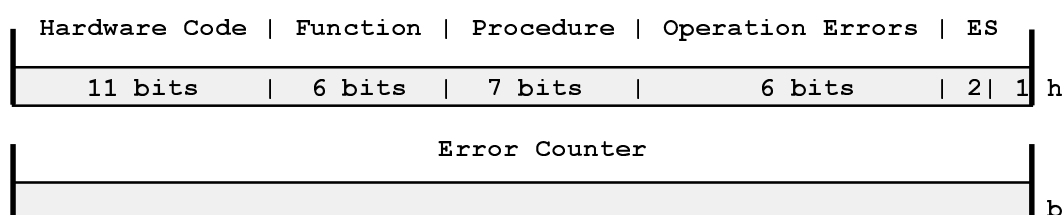
The current principle of DCS functionality (in the construction phase) is "to execute until the first Error" on the any hierarchical level except "periodical" and Agents closing actions. The System Error register contains the unified fields for Function, Procedure and Operation errors and for general Error status:

```

Error Bits 0,1 - Severity of Error : 0 - Normal ("No Error" is even always),
                                     2 - Warning,
                                     1 - Alarm (    Error is odd always),
                                     3 - Fatal;

```

If the system is working in the "periodical" or closing mode the Error counter may be used also. The Error register ERRc (ERR-code) and Error counter ERRn (ERR-number) have 32-bits length and ERRc has following bits distribution:



where ES is Error Severity.

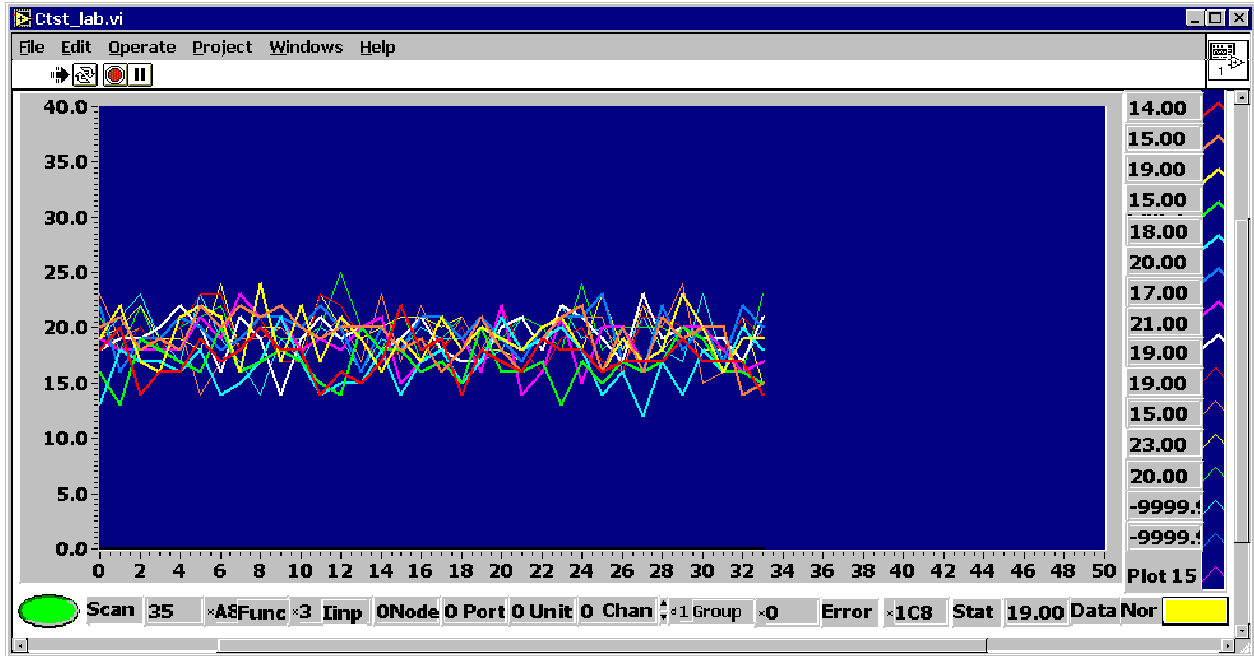
There are no Error printing (drawing) facilities in the DCS Functions themselves (because they are on a "Far side"). The special set of subroutines was developed for the Errors indication in the Text or Graphics modes (on the Client/User side). Some examples of the Error Messages are shown below:

Function Errors:	Procedure errors:	Operation Errors:
.....	.....	.....
" InitDCSunit : CanCrNotifBox() failure	Overflow_in_CAN_chip	"
" WriteDCSdata : CanNmtServGlobal() failure	Write_Queue_OverFlow	"
" ReadDCSdata : CanBoxWrite() failure	Data_just_read_was_old	"
" WaitDCSevent : Can't find Unit by Name	CAN_communication_form	"
.....	.....	.....

#### 4. LabView version of SCT DCS software

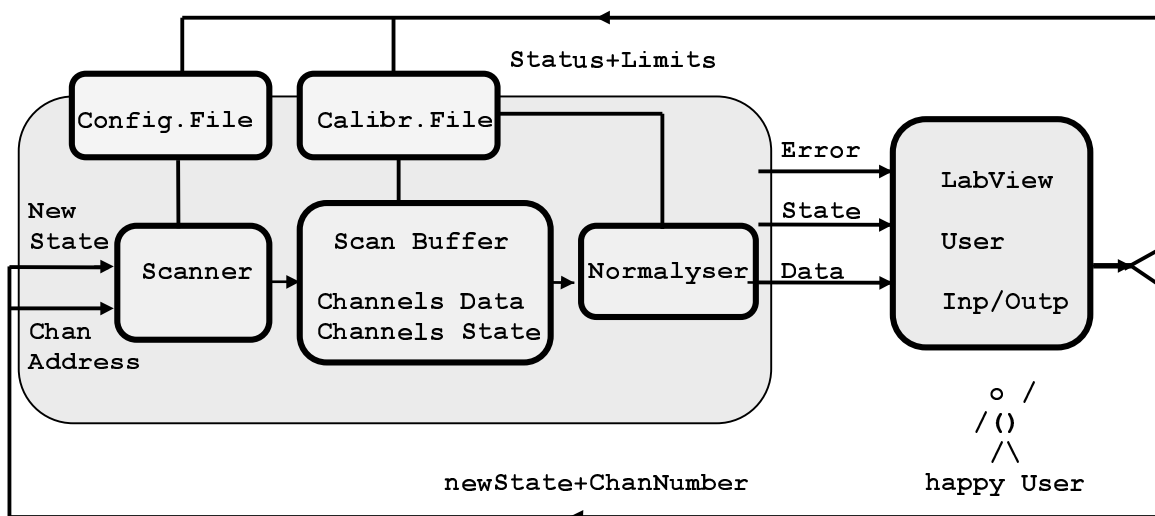
All the core functions and subroutines are written in 'C' but they may be successfully used in the LabView environment as LabView CINI, it is shown in Fig.11.

Figure 11



The DCS Scanning Program in LabView is based on the SetGetByAddress() subroutine and SetDCSState(Scod) system function is called on the every Scan. The “functional box” (left green case in Fig.12) has 6 inputs: 5 - for Host, Node, Port, Unit, Chan numbers; Iinf (input Info) - for the Request and 3 outputs: Dout - for calibrated (or normalised) data Getting; Sout – for current system state; Eout – for an error. The current State and Error information is presented in according with unified Status (p.2.5) and Error (p.3.4) codes convention (the Data are comparing with the Limits).

Figure 12



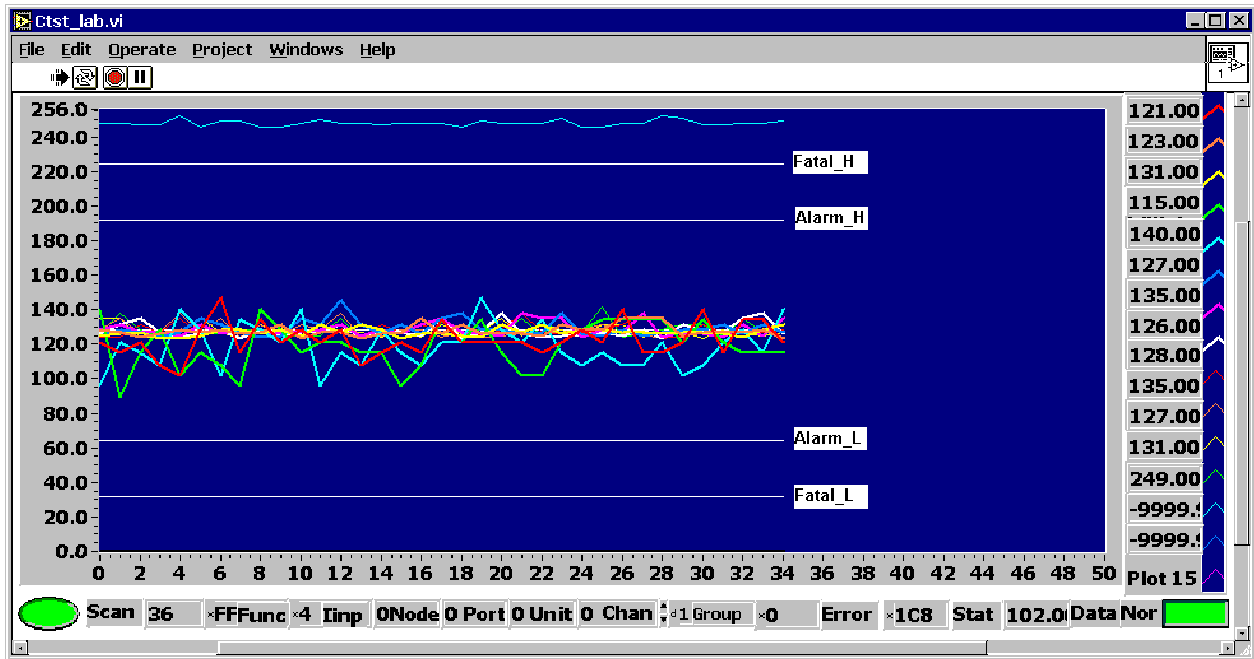
If Func=SETdcsSTATE and Iinf=4Ch the program is scanning one time all the Sensors (of all the Units, of all the Ports, of all the Nodes, of all the Hosts) that are included in the Configure Table, and puts Data and Status values into the DCS-message.

If Func=GETdcsINFO the full Data & Stat message is sending to “near side”.

If Func=COPYgrpINFO the program is making a copy of Chan info that is numbered in ‘Iinf’ (the Group is pointed in ‘Chan’ section of Address).

If Func=COPYchaINFO the addressed Chan Info is copying.

Figure 13



The LabView application is operating in simulation mode in both illustration pictures; in Fig.10 Data are represented as Physical values (room temperatures here). In Fig.13 the “temperatures” are shown as NORMAlized values (right control button is green), in this case the Alarm\_L/H and Fatal\_L/H horizontal lines are generating automatically. As may be easy seen from Fig.13 the last channel is in Fatal\_H zone permanently.

The demonstration LabView application has 2 control buttons only (Scan and Norm) due to it the “Csct\_gr1-4” files are used for other screen settings, the group itself may be selected with the spin-button.

## 5. PVSS application

The PVSS application has a name: “ProjNN” where “NN” is the *project* number. All the “input” (configuration/calibration) files should be put in “Path...\ProjNN\panels” folder; all the “output” files will be created in this folder also.

### 5.1. Indicator & Control Panels and Objects

The main window of the PVSS application is shown in Fig.14. The window contains the following Controls and Indicators.

*System Status Control* (4 buttons in the center below); the system may be in the

- Off state (exit on press);
- Ready state (is using for manual control now);
- Active state, the monitoring is possible here in one of 2 modes:
  - *direct* mode, where “C”-core software is using for Data monitoring (“D” sub-button),
  - *network* mode, where OPC server is sending Data to PVSS application (“Net” sub-button);

One additional button is using for Reloading all the settings from the Config files if something in the system settings was changed.

*Agents Stat/Data Control* (PVSS Text Field from the right below - for password entering) is using for further access to manual control of the system Agents.

*System Action Indicator* (PVSS Text Filed) is placed on the top of agents control field. It shows what system is doing now (see Fig.14,15,16); if system is performing some non-interruptible action than indicator is Yellow.

*Error Indicator* (PVSS Text Field from the left below) supports the error handling mechanism that was described in p.3.4 (see Fig.18 for explanation of the error message created for demonstration).

*Scan Indicator* in the right upper corner of main window, it shows the number of monitoring Scans were done (in Scan mode). It is green usually but while Scan is in the progress it has a yellow color (see Fig.15).

Figure 14



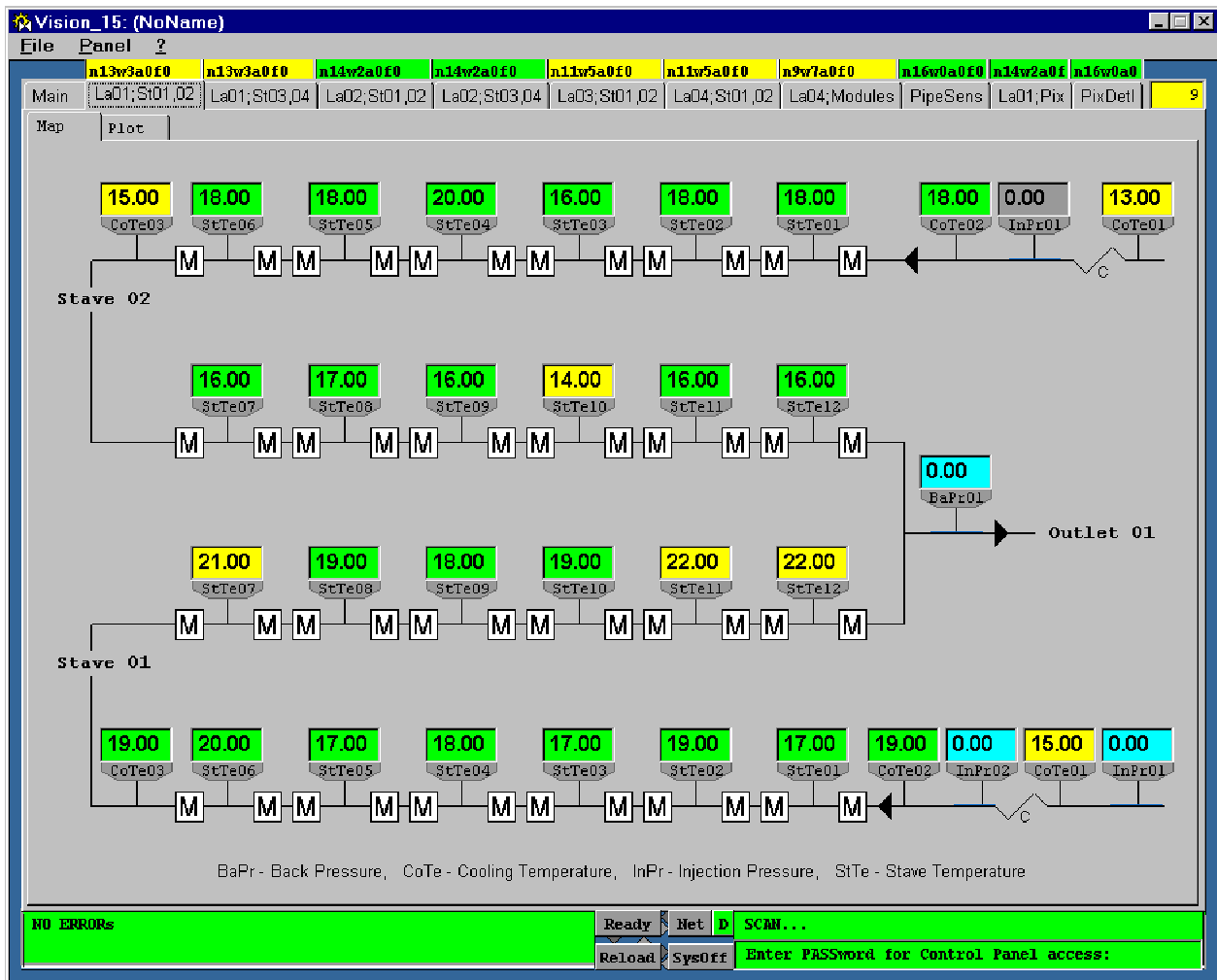
The manual selector in the form of PVSS Tab is included in main window. A Tab contains the Main panel (selected in left top corner of Tab in Fig.14) and 10 watching panels for the Groups of sensors.

Main panel in its turn contains 4 indicators for ambient temperatures and humidity and the Query panel with archive Data extractor “by Name” (p.2.1). The query Name may be individual or broadcast one as it is shown in Fig.14 (4 CAN addresses were found for CoolTemp sensors). The Start and Stop Year-Month-Day-Hour-Minute dates for archive search should be pointed also. After it the non-zero number of query file should be chosen (“1” in Fig.14) and on File-

button click the search is starting. Because a search may take some indefinite time the special indicator in the top right corner shows how many channels are written in the file already. The search results will be written in the query file: “Cout\_guer\_01.dat” where “01” is selected file number in the File spin-button.

Every watching panel in the Tab contains 2 sub-Tabs – one for the sensors schematic “Map” - Fig.15 and another one for the “Plot” (graphical curves, see Fig.16 below).

Figure 15



Hierarchical Status Indicators. The indicators in Fig.15 reflect DCS components status on the 3 levels:

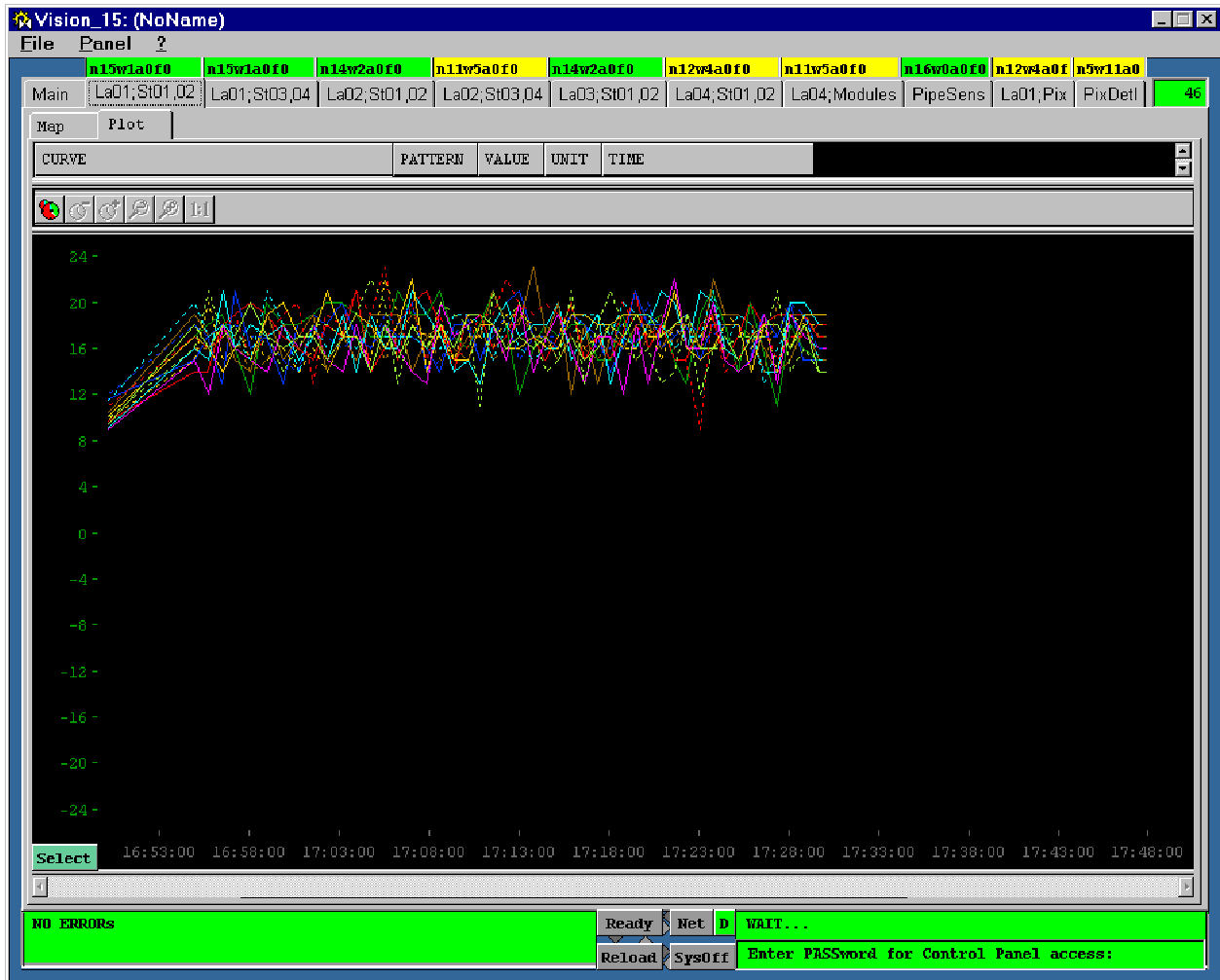
- *sensor status* is represented by the color of sensor indicator, in according with p.1.3 it maybe Green/Yellow/Red/Cyan, if sensor is excluded from the system configuration (its *state* is Off) than the indicator has a Gray color (InjePres01 in Fig15);
- *group status* indicators are placed on the top of appropriate selectors of the Tab (5 are Green and 5 are Yellow), the majority logic is using for Group status determination (the indicator of La01;St01,02 is Yellow because more than 3 sensors have the Warning status, the Fatal – “Cyan” statuses are ignored in this particular case), the numbers of Normal/Warning/Alarm/Fatal sensors are shown in the Group indicator as a short text string;
- *system status* is visible because every Status Control button has a color (“S” button is Green in Fig.15).

Sensor Indicator. This indicator is a reference panel that may be connected to the appropriate DCS channel by using PVSS “dollar parameters” (short form a Name). Two capital

letters with adjacent small letters and an Index are extracting from a short Name and are using as a sensor *label* (abbreviation) that may be read below indication window. For instance, “CoTe03” (upper left corner) is formed from *CoolTemp03*.

The PVSS Trend panel is used for a Plot, it is shown in Fig.16 (PVSS application here is working in simulation mode). The panel has a manual selector also (left down corner) for adding the channels in a Plot “by Name”.

Figure 16



The Group selection panel is shown in Fig.17. This panel consists of 2 parts, one is a Group Selector/Indicator and another one is extractor of archive Data for that Group. The first (upper) part contains:

- *Croup Indicator* in the left upper corner (it is the number of sub-Tab);
- *Curve Number* (Control, spin-button in the left down corner) selects the number of a line;
- *Name Selector* (Control) with the Enter button;
- *Names/Address Indicator* of the sensors/channels that are included in the watching list, the color and type (solid/dash) of the curve for selected channel is shown from the left and between them is
- *Sensor Indicator* (the same as in the Map) for selected sensors/channels;
- *Close Button* (Control).

WARNING: the PVSS Combo-Boxes and Spin-Buttons “remember” the previous entered value but the first “field” and “zero” *are visible* on a panel “pop-up”, it means that *every* parameter should be re-selected on panel opening.

If the wrong Name will be entered in the Name Selector than the following Address will be printed in the Address Indicator section: n”\_p”\_Unit99\_Chann99.

Figure 17

Curve	Group	NAME	ADDR
01	2	Lev5.Lev4.Lev3.Lev2.Lev1	Host.Node.Port.Unit.Chan
02	2	SCT.Layer01.Loop01.Stave01.CoolTemp01	n0_p0_Unit01.Chan00
03	2	SCT.Layer01.Loop01.Stave01.CoolTemp02	n0_p0_Unit01.Chan01
04	2	SCT.Layer01.Loop01.Stave01.CoolTemp03	n0_p0_Unit01.Chan02
05	2	SCT.Layer01.Loop01.Stave01.StavTemp01	n0_p0_Unit02.Chan32
06	2	SCT.Layer01.Loop01.Stave01.StavTemp02	n0_p0_Unit02.Chan33
07	2	SCT.Layer01.Loop01.Stave01.StavTemp03	n0_p0_Unit02.Chan34
08	2	SCT.Layer01.Loop01.Stave01.StavTemp04	n0_p0_Unit02.Chan35
09	2	SCT.Layer01.Loop01.Stave01.StavTemp05	n0_p0_Unit02.Chan36
10	2	SCT.Layer01.Loop01.Stave02.CoolTemp01	n0_p0_Unit01.Chan04
11	2	SCT.Layer01.Loop01.Stave02.CoolTemp02	n0_p0_Unit01.Chan05
12	2	SCT.Layer01.Loop01.Stave02.CoolTemp03	n0_p0_Unit01.Chan06
13	2	SCT.Layer01.Loop01.Stave02.StavTemp01	n0_p0_Unit02.Chan44
14	2	SCT.Layer01.Loop01.Stave02.StavTemp02	n0_p0_Unit02.Chan45
15	2	SCT.Layer01.Loop01.Stave02.StavTemp03	n0_p0_Unit02.Chan46
16	2	SCT.Layer01.Loop01.Stave02.StavTemp04	n0_p0_Unit02.Chan47
17	2	SCT.Layer01.Loop01.Stave02.StavTemp05	n0_p0_Unit02.Chan16

0 SCT AmbiHumi 0 Loop 0 Stave 0 BackPres 0 Enter

year month day hour minutes Stop year month day hour minutes Start

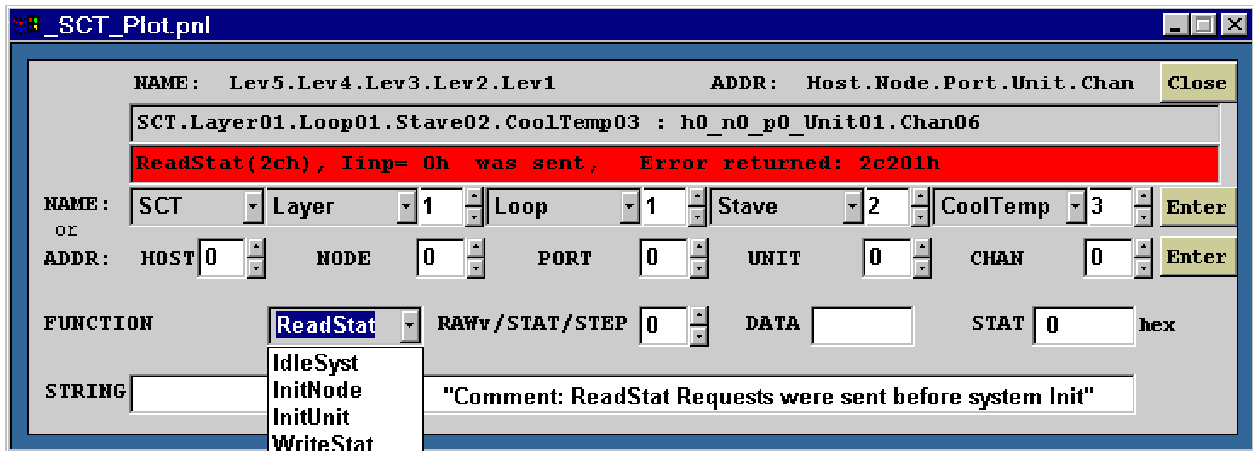
2002 2 16 20 6 Now 2001 1 1 0 0 Now File 1

The Group archive Data extractor is practically the same as was described in Fig.14, the search results will be written in the group file: “Cout\_grup\_NN.dat” where “NN” is non-zero number.

The Control panel is represented in Fig.18, it is opening on the password entering in Agents Stat/Data Control text field that was shown in Fig.14 (down right corner). By using this panel every Function from a system set (Table 2) may be requested manually in Scan mode – see the part of a Function list in the combo-box. The identification of desirable sensor/channel may be done via the Name selector or via Address selector as well (2 Enter buttons). The returned Physical Data or State (hexadecimal code) will be shown in their indicators. The input value (it may be Raw value, State or RunScript “step” number) is entered with the help of a spin-button (at center).

There is a possibility to send/take a string via PC serial port; the string may be entered (and taken string may be indicated) in the String Text Field (below). The info exchange with the PC parallel port may be done in usual way. The panel may be closed by click on Close button.

Figure 18

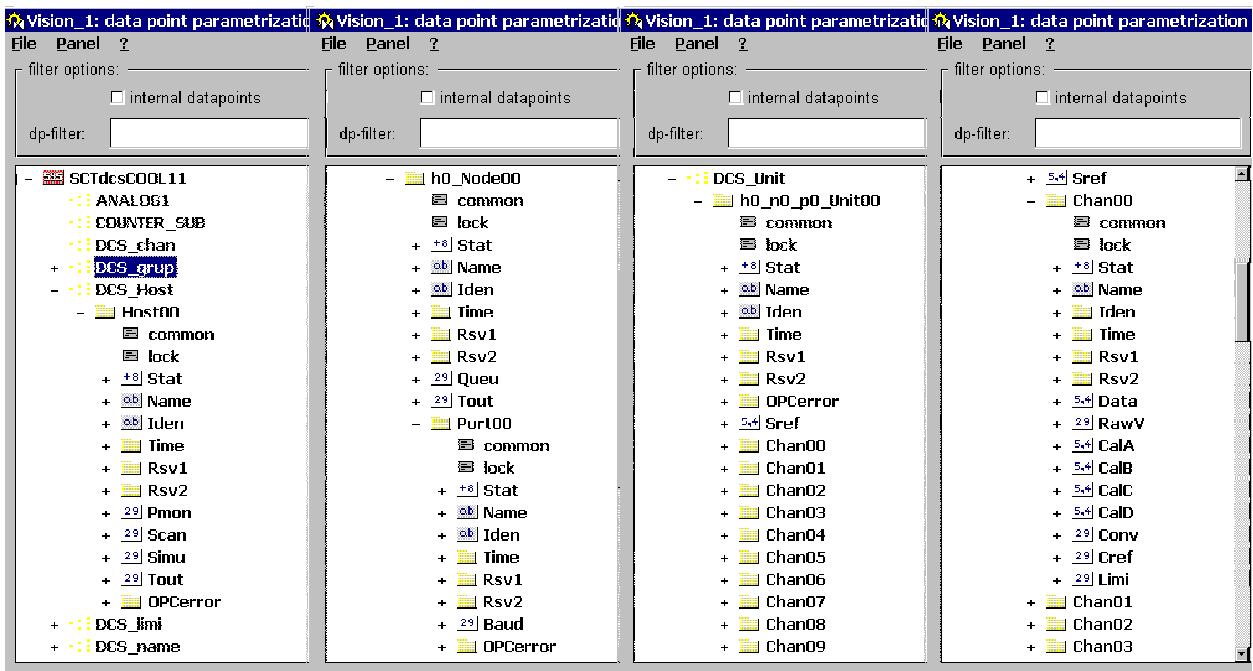


The Fig.18 illustrates the attempt of reading status of CoolTemp03 sensor *before* the system initialization. In the case of error the details of error info appear in the Error Indicator of the main window (see Fig.14). The recommended system state for manual Control is Ready (but it may be done in the Active state as well) in Scan mode.

## 5.2. PVSS Data Points structures

The main elements of PVSS Data Points (DP) are similar to the elements of DCS Data Base. The DP-elements for DCS Agents (Host, Node+Port, Unit, Chan) are represented in Fig.19.

Figure 19



The common DP-elements for all the Agents are the following (4-letters names are used):



### 5.2.1. Data Points loading from the Data Base

If something in the system settings was changed the DP-elements should be reloaded by click on the Reload button in System State Control (see Fig.14). The DCS system should be in the Off or in the Ready states in this case.

The configuration file: "OPCCanServer.Cfg" for OPC server is creating in Reload mode, it should be placed in the OPC server folder (see appropriate manual). The following files are generated additionally during this process: "OPCini.bak", "OPCpdi.bak", "OPCpdo.bak" and "OPCsdo.bak", they may be deleted.

### 5.3. PVSS Calls of "C"-functions

For increasing the PVSS code productivity the mostly repeated DCS actions are making via the calls of "C"-core subroutines (see p.3). The full set of System Functions and libraries of "C"-core code is widely used in the Direct mode.

For Network mode (OPC server) the following "C"-subroutines are using only (no other libraries):

```
/* THE "C" FUNCTION FOR RANDOM INTEGER GENERATING */
/*
/*          Zone - random values range
/*          Form - FORM of distrib. curve :
/*                  Form=0 - (Shif+(Zone/2)) peak
/*                  Form=1 - Flat from Zero to Zone
/*                  Form=2 - Triangular form
/*                  Form>>1 - Narrow Gaussian curve
/*          Shif - SHIFt of the distribution center
/*
int RandomInteger(int Zone, int Form, int Shif);
```

```
/* THE "C" FUNCTION FOR DCS ADDRESS GETTING BY USING SENSOR/CHAN NAME */
/*
/*          Name - NAME of the Sensor/Chan
/*          .... - ....
/*          Return - Error code
/*
int GetAddressByName(char *Name, int *Host, int *Node, int *Port,
                    int *Unit, int *Chan);
```

```
/* THE "C" FUNCTION FOR SENSOR/CHAN NAME GETTING BY USING DCS ADDRESS */
/*
/*          .... - ....
/*          Name - NAME of the Sensor/Chan
/*          Return - Error code
/*
int GetNameByAddress(int Host, int Node, int Port,
                    int Unit, int Chan, char *Name);
```

```
/* THE "C" FUNCTION FOR SQL ADDRESS LIST CREATING BY USING "BROAD" NAME */
/*
/*          Name - Broadcall NAME of Sensor(s)/Chan(s) in SQL
/*          Leng - LENGth of Address List created
/*          Return - Error code
/*
int ListAddressesByBroadName(char *Name, int *Leng);
```

```

/* THE "C" FUNCTION FOR TAKING NUMBERED ADDRESS FROM SQL LIST          */
/*                                                                      */
/*          Numb - NUMBer of Address in SQL List                      */
/*          .... - ....                                              */
/*          Return - Error code                                       */
/*                                                                      */

int GetAddressFromList(int Numb, int *Host, int *Node, int *Port,
                      int *Unit, int *Chan);

```

```

/* CONVERT RAW VALUE TO PHYSICAL DATA AND COMPARE IT WITH THE LIMITS */
/*                                                                      */
/*          Raw0 - RAW Value from current sensor                      */
/*          Raw1 - RAW Value from the first (humidity) sensor        */
/*          Conv - CONVersion formula number                         */
/*          Sref - Sensor REference (bias voltage)                   */
/*          Cal* - CALibration constants A,B,C,D                     */
/*          Limi - LIMIt type                                         */
/*          Data - Physical calibrated DATA                          */
/*          Stus - STATUS (Error Severity) of a Chan(nel)            */
/*          Return - Error code                                       */
/*                                                                      */

int RawToPhysLimits(int Raw0, int Raw1, int Conv, float Sref,
                    float CalA, float CalB, float CalC, float CalD,
                    int Limi, float *Data, int *Stus);

```

All these subroutines are placed in “DllOrigin.c” file (the first DLL-level in “C”). The second DLL-level is represented by “DllPvss.cpp” library (“C++” is necessary for PVSS) it is just an intermediate (transparent) layer between PVSS and “C”-code.

## 6. Calibration software

There are the following possible sources of Physical Data tolerances (in increasing order):

- the differences in the sensors parameters,
- the differences in ADCs offsets (pedestal) and conversion factors (gain),
- the differences in the “bias current” resistors (or divider resistors) for sensors.

It is supposed that bias and divider resistors (Adapters) are the part of electronics, it means that their differences should be compensated by “A” and “B” calibration constants.

For SCT DCS temperature monitoring the precision requirements are not so high (~ 0.5 °C). In this case the parameters of sensors themselves are practically identical (in one delivery) and “C” and “D” calibration constants may be unified for definite sensor type.

The calibration procedure for ELMBs analogue inputs will be briefly described below; for other types of CAN-boxes it may be done in a similar way. The calibration procedure is making in 2 steps:

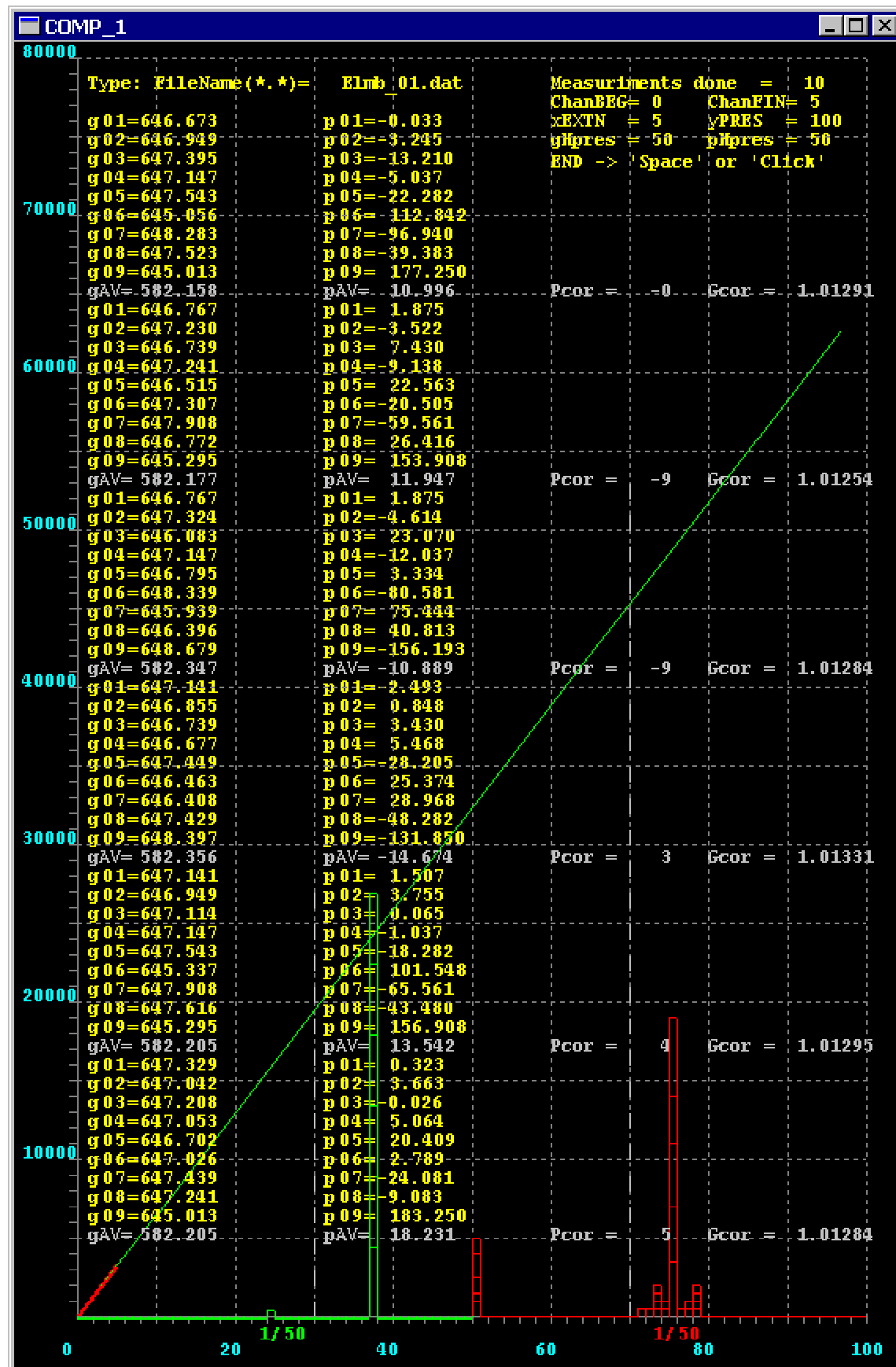
- intermediate calibration of ADCs (the first correction of “A” and “B” is on the output);
- calibration of Adapter resistors and finding the final values for “A” and “B”.

The first calibration should be done before plugging the "bias current" Adapters (the original “1k” direct connected resistors should be used for this test). All the ELMBs inputs are connecting in parallel and 10 voltage steps (say 1, 11, 22, 33, ..., 99 mV) are generating in sequence. The Data for every step from all the channels are taking by "Ctst\_txt.exe" program (Manual Controller of CAN-bus, see p.7 below) and are writing in “Cout\_inf.dat” file. After it "GainPedestHist.exe" program is starting for watching of measurement results and for “A” & “B” correction values finding. The example of executing of this program is shown in Fig.21. The following info may be seen on the screen:

- "step by step" measurements of transfer characteristics of selected channels (Yellow text);
- the linearity curves from that channels (Green lines);

- the projections of the every step of the every line onto Y-axis (i.e. the "pedestals" - short Red lines from the left);

Figure 21



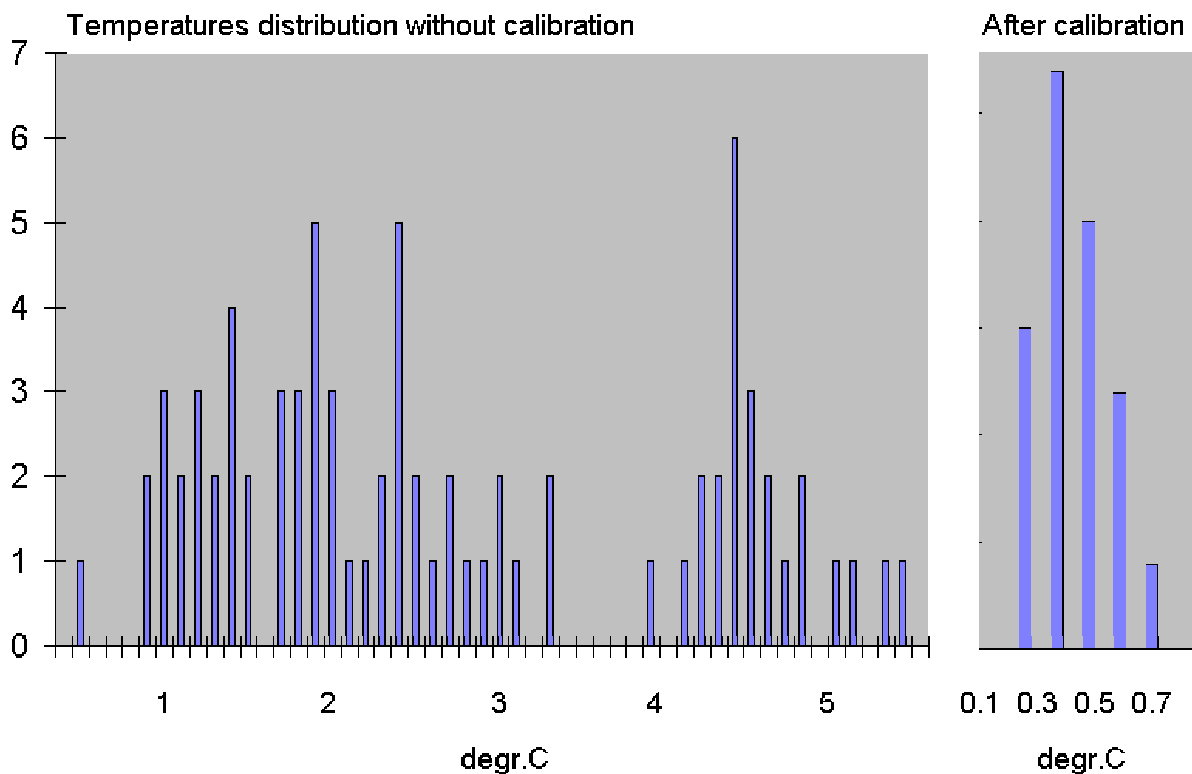
- the correction values for ELMB channel Gain and Pedestal that were taking between 30 mV and 70 mV of input voltages (Gray text).

The second step may be done after plugging the "bias current" Adapters into ELMBs and plugging input connector with 16 resistors (metal film) that “replace” the sensors, i.e. their nominal values

- 1 kOhm for PT1000,
- 10 kOhm for NTC inputs.

Every of 16 resistors should be measured individually before and its real value should be written in the proper column of “ELMBsens.dat” File. The "Ctst\_txt.exe" is starting 4 times because the set of calibrated resistors is connecting serially to other input connectors of ELMB. The final calibration values for “A” and “B” are calculated by "GainWithAdapt.exe" program and may be found in “Elmb\_NNcal.dat” file.

Figure 22



In Fig.22 are shown “channel to channel” distribution of Temperature values before and after calibration (the measurements were done by P-O.Wallin).

## 7. The Testing and Simulation software

Testing software is written in “C” and based on “C”-core subroutines (p.3). The same configuration files (as for main application) are necessary for it running. Testing software may be used in 2 ways:

- instead of PVSS (LabView) applications for checking CAN hardware state and status;
- in parallel with main application from the second port of CAN-controller card, in this case

CAN-bus should be connected to the both connectors of controller card.

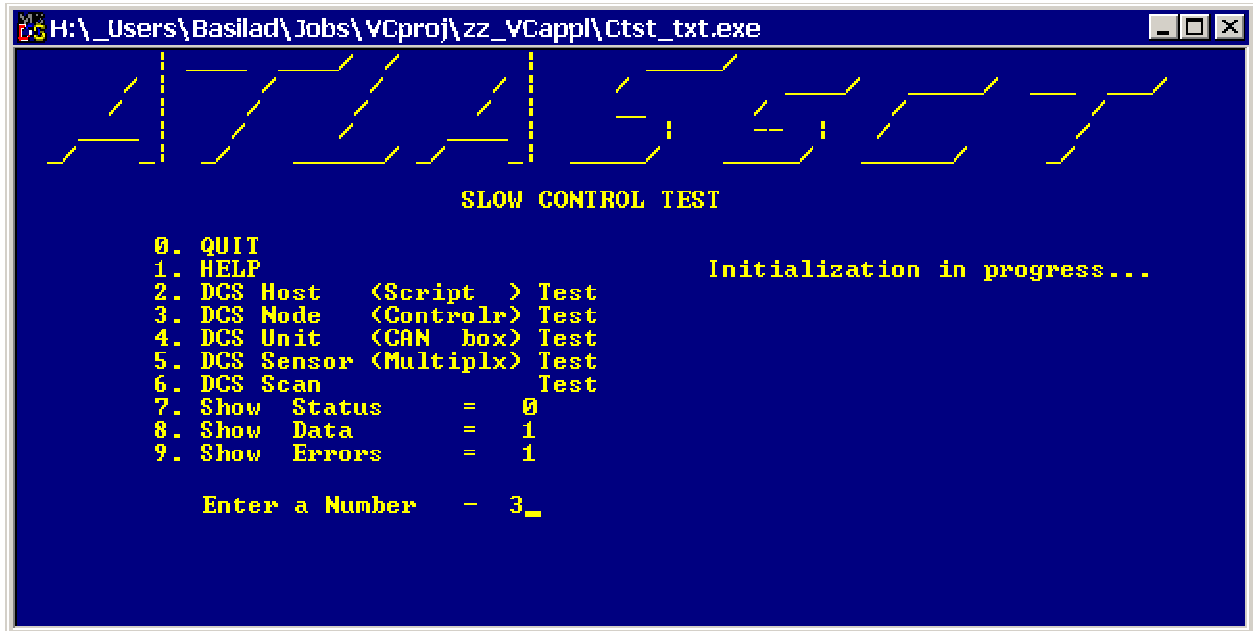
There are 4 reasons to use a special designed software (in “C”) for testing instead of main (PVSS) application:

- it is much more faster;
- it is much more simple in use;
- CAN-data taking process is visible;
- every software step may be checked with the powerful Debugger (Visual C, Borland C).

## 7.1. Manual Controller in “C”

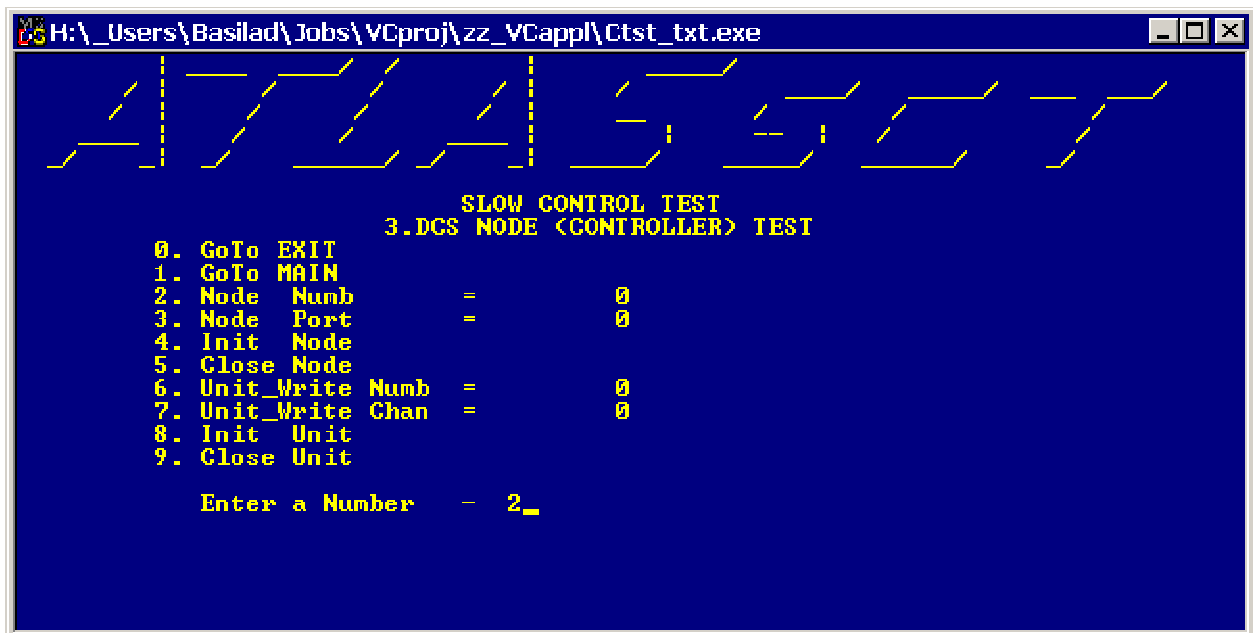
The first testing and diagnostics program is “Ctst\_txt.c”, it provides all the System Functions (p.3.1.1) and supports a User interface in the text mode. The menu of the first level is shown in Fig.23.

Figure 23



If Host Test (2) or Node Test (3 – see Fig.23) or Unit Test (4) was selected than system initialization will be done automatically (it is shown in Fig.23). In the Node Test mode – see Fig.24, the desirable Node number or Port number or Unit or Chan number for writing may be defined.

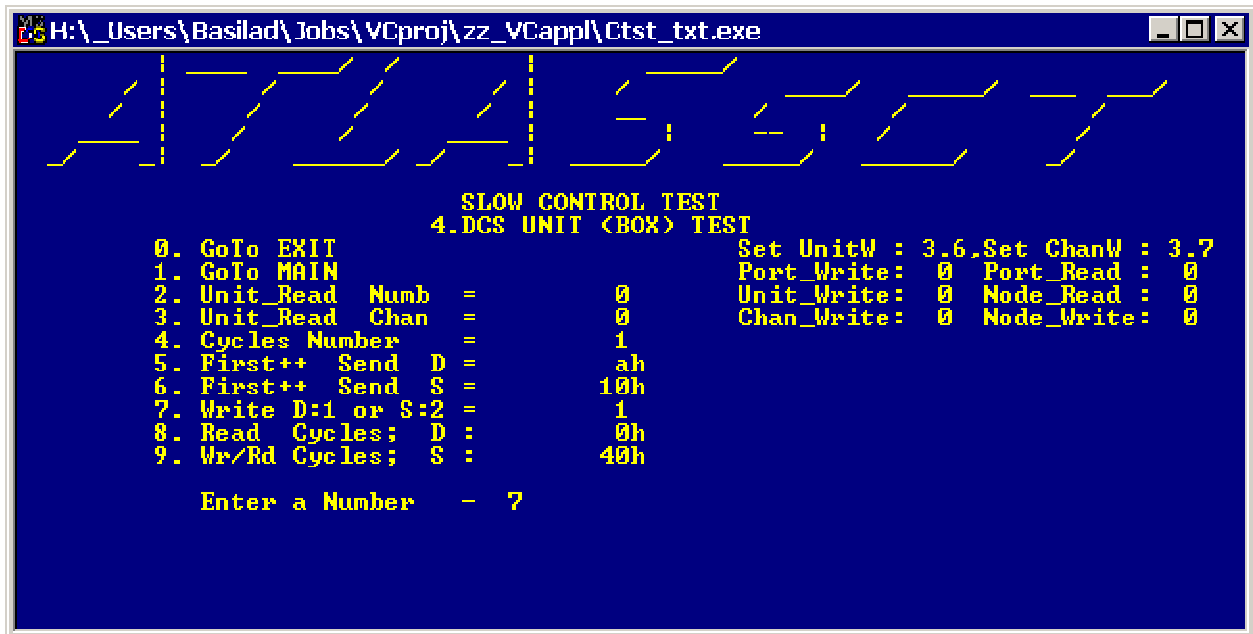
Figure 24



If Unit Test mode was selected – Fig.25 than Data or State may be sent (to) or taken (from) any channel of any Unit. The writing Info may be defined via “5” (Data) or “6” (State) and “7” is

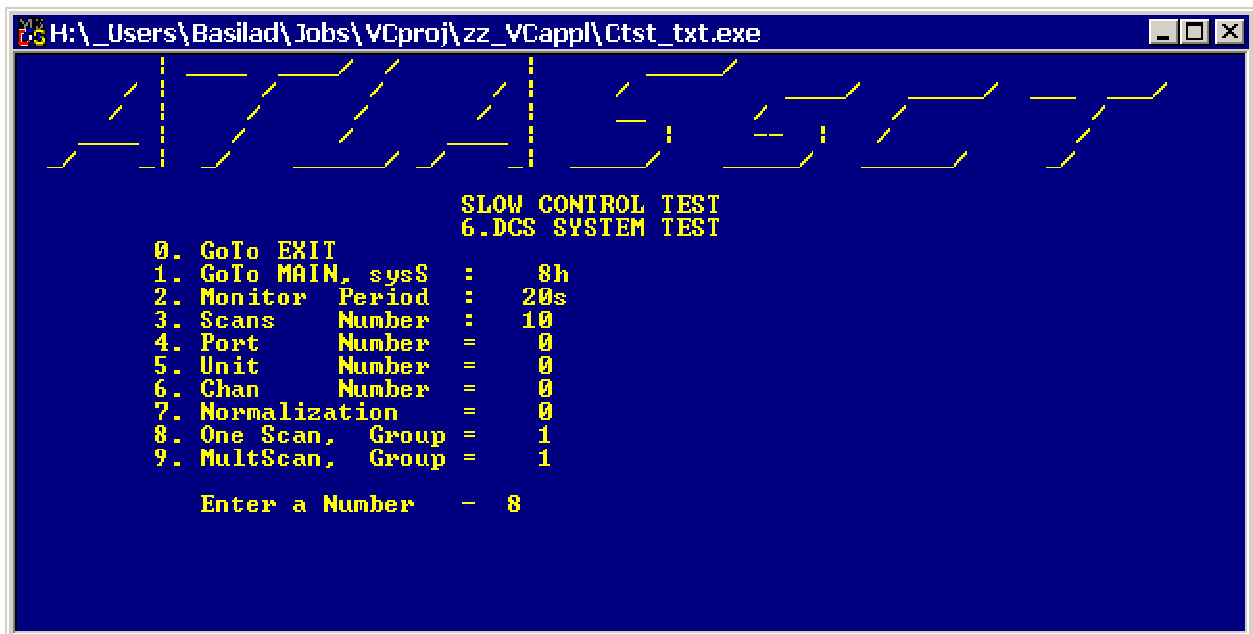
selector for Data ("1") or State ("2") writing. The reading is making by choosing of "8" and read-then-write cycles are selected by "9". The number of cycles is defined via "4".

Figure 25



The next is so-called System Test ("6" in main menu) – Fig.26. The scanning of all the (included in System configuration) Units is making in this mode. This mode is using for calibration procedures.

Figure 26



Two system operations are possible here: one Scan ("8") or multiScan ("9"). The number of scans ("3") and period of monitoring ("2") in the second case are reading from the Host Config file. In both cases the watching Group number may be selected (2 digits are typing: "8" and then "GrupNumb" or "9" and then "GrupNumb" – 1 in Fig.26).

During the program execution in these modes the Raw values taking process is visible – see Fig.27 on the top. The 1-st and the 3-rd lines show Unit-Chan numbers; the 2-nd and the 4-th lines show the Raw values in these Chan-s.

Figure 27

```

H:\_Users\Basilad\Jobs\VCproj\zz_VCappl\_Ctst_txt.exe
-31 6-32 6-33 6-34 6-35 6-36 6-37 6-38 6-39 6-40 6-41 6-42 6-43 6-44 6-45 6-46
b5fc 00a8 d9b6 ae46 6672 2b1 132e 00 00 008 1df5 693 b718 4c43 13ff ffff
-47 6-48 6-49 6-50 6-51 6-52 6-53 6-54 6-55 6-56 6-57 6-58 6-59 6-60 6-62 6-63
766e 003 11db 0066 006b 00 132f 009 00 00 18e9 241 00 00 00a 37b3

SLOW CONTROL TEST
6.DCS SYSTEM TEST

0. GoTo EXIT
1. GoTo MAIN, sysS : 48h
2. Monitor Period : 20s A 21.2 A 18.8 A 17.9 A 19.9 F-257.5
3. Scans Number : 10
4. Port Number = 0
5. Unit Number = 0
6. Chan Number = 0
7. Normalization = 0
8. One Scan, Group = 1
9. MultScan, Group = 1

Enter a Number - 8_

```

The first 5 monitored Chan-s of the watching group are indicating (status – N,W,A,F and Physical Data) on the right side of the text panel.

## 7.2. CAN-bus analyzer

The second program “NIsnoop.c” is based on National Instruments code for “CAN-analyzer” – Fig.28.

Figure 28

```

H:\_Users\Basilad\Jobs\VCproj\zz_VCappl\_NIsnoop.exe

NI-CAN ANALYSER: Type SnoopPort = 1, BaudRate(kBod) = 250, TimeOut(sec) = 59

T: 50m:06s, R: 80567, I: 287h, L: 4, D: 2dh 11h dfh 18h U= 7 C=45 D= 6367
I: 50m:06s, R: 83540, I: 287h, L: 4, D: 2eh 91h ffh ffh U= 7 C=46 D=65535
T: 50m:06s, R: 86512, I: 287h, L: 4, D: 2fh 11h 91h 8bh U= 7 C=47 D=35729
I: 50m:06s, R: 89216, I: 287h, L: 4, D: 30h 91h 0h 0h U= 7 C=48 D= 0
T: 50m:06s, R: 92061, I: 287h, L: 4, D: 31h 11h 32h 14h U= 7 C=49 D= 5170
I: 50m:06s, R: 95071, I: 287h, L: 4, D: 32h 91h 0h 0h U= 7 C=50 D= 0
T: 50m:06s, R: 96473, I: 287h, L: 4, D: 33h 91h 0h 0h U= 7 C=51 D= 0
I: 50m:06s, R: 97583, I: 287h, L: 4, D: 34h 91h 0h 0h U= 7 C=52 D= 0
T: 50m:06s, R: 0, I: 287h, L: 4, D: 35h 11h 2dh 15h U= 7 C=53 D= 5421
I: 50m:06s, R: 1853, I: 287h, L: 4, D: 36h 91h 0h 0h U= 7 C=54 D= 0
T: 50m:07s, R: 4497, I: 287h, L: 4, D: 37h 91h 0h 0h U= 7 C=55 D= 0
I: 50m:07s, R: 4501, I: 287h, L: 4, D: 38h 91h 0h 0h U= 7 C=56 D= 0
T: 50m:07s, R: 6806, I: 287h, L: 4, D: 39h 11h 8dh 1bh U= 7 C=57 D= 7053
I: 50m:07s, R: 9829, I: 287h, L: 4, D: 3ah 11h dfh 1h U= 7 C=58 D= 479
T: 50m:07s, R: 12580, I: 287h, L: 4, D: 3bh 91h 0h 0h U= 7 C=59 D= 0
I: 50m:07s, R: 12581, I: 287h, L: 4, D: 3ch 11h c1h 1h U= 7 C=60 D= 449
T: 50m:07s, R: 15374, I: 287h, L: 4, D: 3dh 11h 62h 14h U= 7 C=61 D= 5218
I: 50m:07s, R: 18441, I: 287h, L: 4, D: 3eh 91h 0h 0h U= 7 C=62 D= 0
T: 50m:07s, R: 21114, I: 287h, L: 4, D: 3fh 11h 73h 37h U= 7 C=63 D=14195

```

It should be started from another CAN-port (CAN-bus should be connected to both). Before running the Port number have to be defined as well as BaudRate and TimeOut values.

CAN-analyzer shows:

- time of CAN-frame appearing on the CAN-bus (T);
- number of waiting loops till the current frame (R);
- COB-ID in the frame (I);
- length of the message info (L) in bytes;
- Raw data bytes (D) – 4 for ELMB;

and then in the last 3 columns: Unit and Chan numbers and Physical Data, i.e. decoded frame information (D).

Some care should be taken while using CAN-analyzer because it may take a significant part of processor resources (waiting loops).

The simulation software. The special set of Data/State Read/Write Init/Close *simulation* Functions was created (“Cfun\_sim.c” library); it allows to exclude CAN hardware from the slow control system for testing of other DCS parts and for software debugging. Any errors can be generated in a random way with desirable probability in Scan mode (needs recompiling), it permits to check the system sensitivity to errors and the system tolerance to them.

For running in the simulation mode the “DataSimu” bit should be installed in a Host Config file.

## 8. Supplements

### 8.1. The list of software libraries and Version code

- Cversion.c - This file contains all the Global "C" variables and structures and controls all the possible Oper\_Modes. It may be Compiled and Run as the Server or as the Client, or as a Library (CIN for the LabView), or as the Test program in the Text or Graphical modes. The proper definitions must be done in Define Modes Area. If CANVHEAD and LIBRaries are defined than the File can be used as a Header File.
- Ctst\_txt.c - CAN modules TeST in the TeXT mode, main program (for expert). Tests all the CAN programs (Open/Close, Write/Read/Event) on the Hardware Independent Trans/Action Level in the Text Terminal mode.
- Ctst\_gra.c - CAN modules TeST in the GRaph mode, main program (for expert). Tests all the CAN programs (Open/Close, Write/Read/Event) on the Hardware Independent Trans/Action Level in the Graph Terminal mode.
- Ccom\_lib.c - This library contains the subroutines for execution of Script described procedures.
- Creq\_lib.c - Provides CAN high level REQuests for the System Initialization, Setting the States and Getting Slow Control Data.
- Cact\_lib.c - CAN ACTions LIBrary. Provides Network (TCP/IP) or Direct Link from Client CAN/VME program to Server CAN/VME Functions that deal with hardware or with CAN/VME driver simulators. May be used as  
a) SIMulation library of network Actions (SIMa);  
b) NETwork Client library (NETC),  
c) NETwork Server program (NETS - in this case the name of this File should be changed to Cact\_ser.c).
- Csys\_lib.c - CAN SYStem LIBrary. Makes the distribution of the calls to the vendor oriented CAN subroutines for the System Host Initialization, Setting the States and Getting the Data from Detector Control System level.
- Cnam\_lib.c - Makes on call: Naming file reading and then Detector names and CAN addresses structures filling; getting CAN Address by Detector Name and getting Name by Address.
- Cerr\_lib.c - CAN ERRror LIBrary. Provides all the necessary Error Handling System subroutines on the Procedure (hardware dependent), Function (hardware independent) and Action (remote control) levels.
- Cfun\_sim.c - CAN FUNctions SIMulation library. Provides al the necessary (Open/Close, Write/Read/Event) simulation subroutines for CAN controllers and boxes on the Function (hardware independent) and Action (remote control) levels.
- Comp\_lib.c - COMPUter COMPatible LIBrary. Creates Platform independent Input-Output Text-Graphics facilities in "C".

Nlca\_lib.c - National Instruments CAN controller LIBrary. Provides all the necessary (Open/Close, Write/Read/Event) subroutines that are based on Nican Controller drivers for the Procedures (hardware dependent level).

Nlmb\_lib.c - National Instruments & local Monitor Box LIBrary. Provides all the necessary (Open/Close, Write/Read/Event) subroutines for Nican drivers and for LMB boxes on the Procedure (hardware dependent), Function (hardware independent) and Action (remote control) levels.

Nlcl\_lib.c - National Instruments & Embedded Local monitor box LIBrary with the same possibilities as Nlmb\_lib.c.

Nlor\_lib.c - National Instruments & ORic box LIBrary. Provides all the necessary (Open/Close, Write/Read/Event) subroutines for NTcan drivers and for CST boxes on the Procedure (hardware dependent), Function (hardware independent) and Action (remote control) levels.

PCsp\_lib.c - The library for IBM\_PC Serial Ports.

PCpp\_lib.c - The library for IBM\_PC Parallel Ports.

All the DCS settings and calibration constants are reading from the proper Files during the Host initialisation. These files are generated from Microsoft ACCESS Data Base Tables:

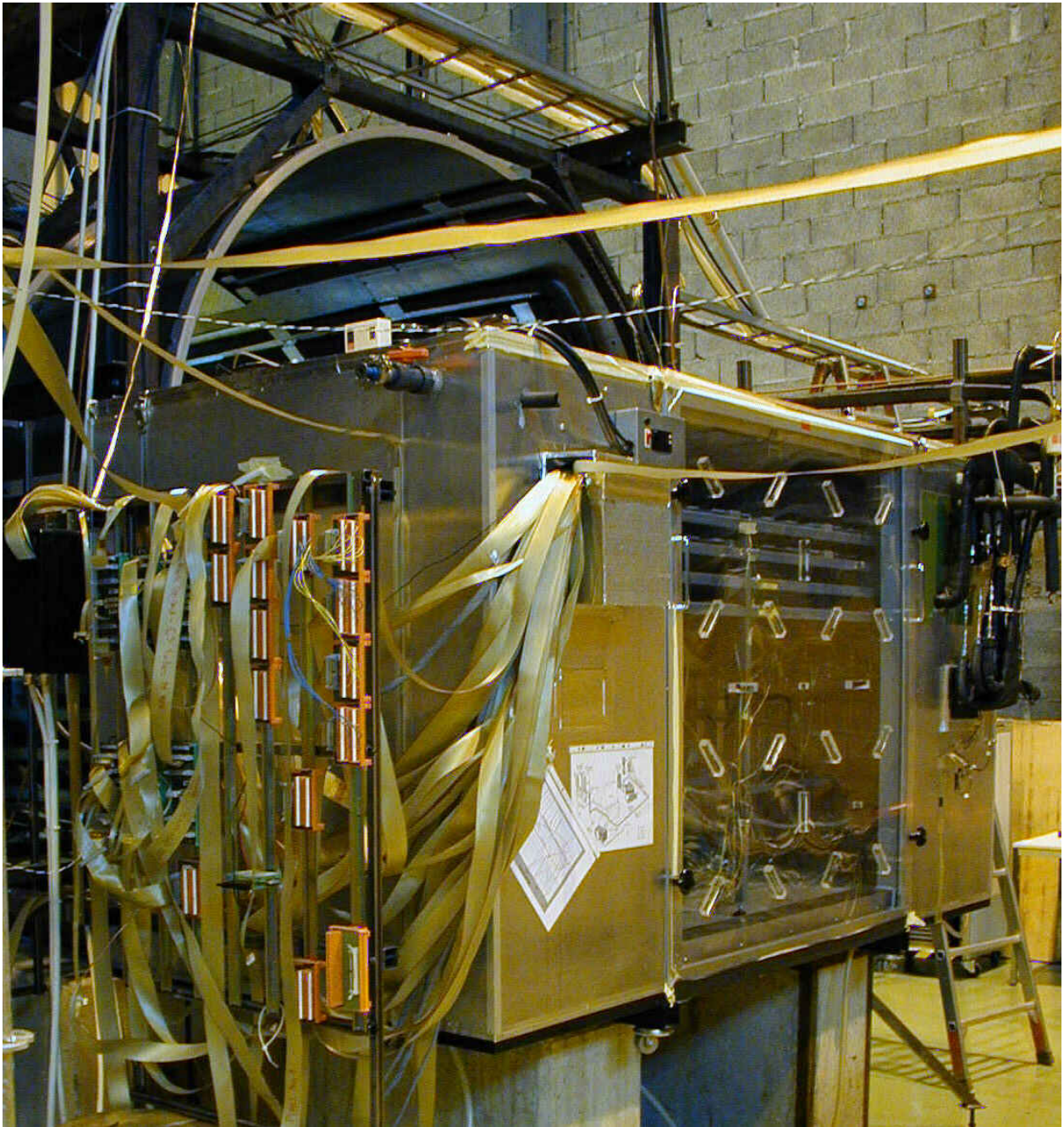
Csct_sys	- CAN (ATLAS) SCT SYSTEM parameters,
Csct_hst	- CAN (ATLAS) SCT HoSTs parameters,
Csct_nod	- CAN (ATLAS) SCT NODEs parameters,
Csct_uni	- CAN (ATLAS) SCT UNITs parameters.
Csct_cnf	- CAN (ATLAS) SCT naming and calibration CoNFig info.
Csct_lim	- CAN (ATLAS) SCT LIMits types parameters.
Csct_gr1-8	- CAN (ATLAS) SCT Scan selection groups.
Csct_ini	- Script that is executed on the end of initialization,
Csct_run	- Script that is executed in the Quick Loop,
Csct_off	- Script that is executed on the start of system closing.

The both (text and graphics) test programs may be used for any combination of the working and simulation libraries. After choosing the necessary combination of libraries (in the Cversion.c file) the proper version code is generating (the first compiler operation). Version code "register" contains 3 fields - Hardware (8 bits), Simulation (4 bits) and Terminal mode (4 bits) linear Codes:

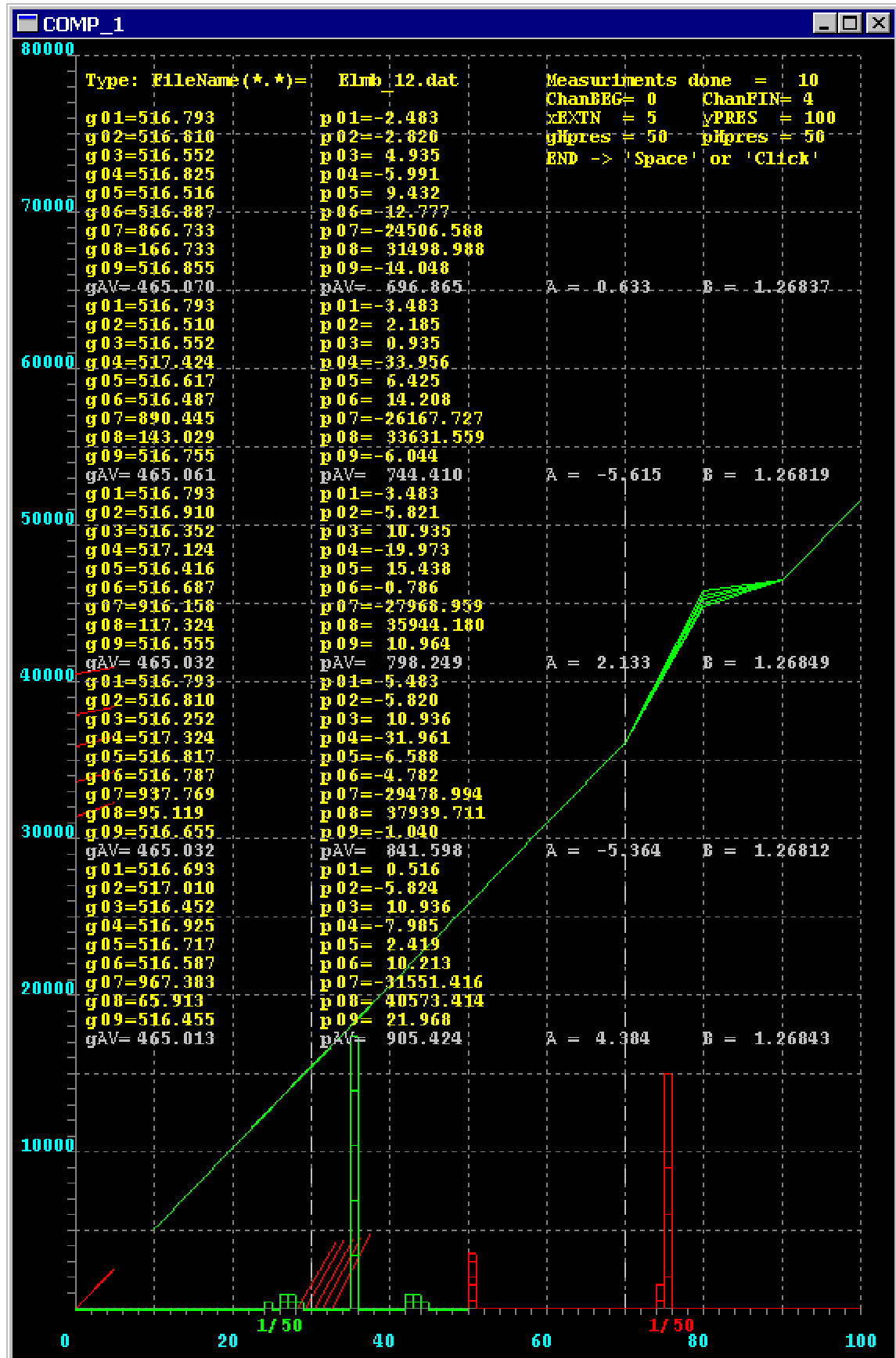
SystCode	HardCode	SimuCode	Term	ShowCode
	see Hardware Version Code	A F D   M	A S D E	
8000 4000 2000 1000 800 400 200 100	80  40  20  10	8  4  2  1 h		

Simulation CODE:		Terminal Code:		Show CODE:	
D - Driver	Simu/Work,	If M=1 -> Text	terminal mode,	E - Errors Show,	
F - Function	Simu/Work,	If M=0 -> Graph	terminal mode;	D - Data Show,	
A - Action	Simu/Work;	System Code: 00 - ATLAS,		S - Status Show,	
		10 - ZEUS;		A - Action Show.	

## 8.2. The general view of DCS-hardware for SCT-Cooling in B.175



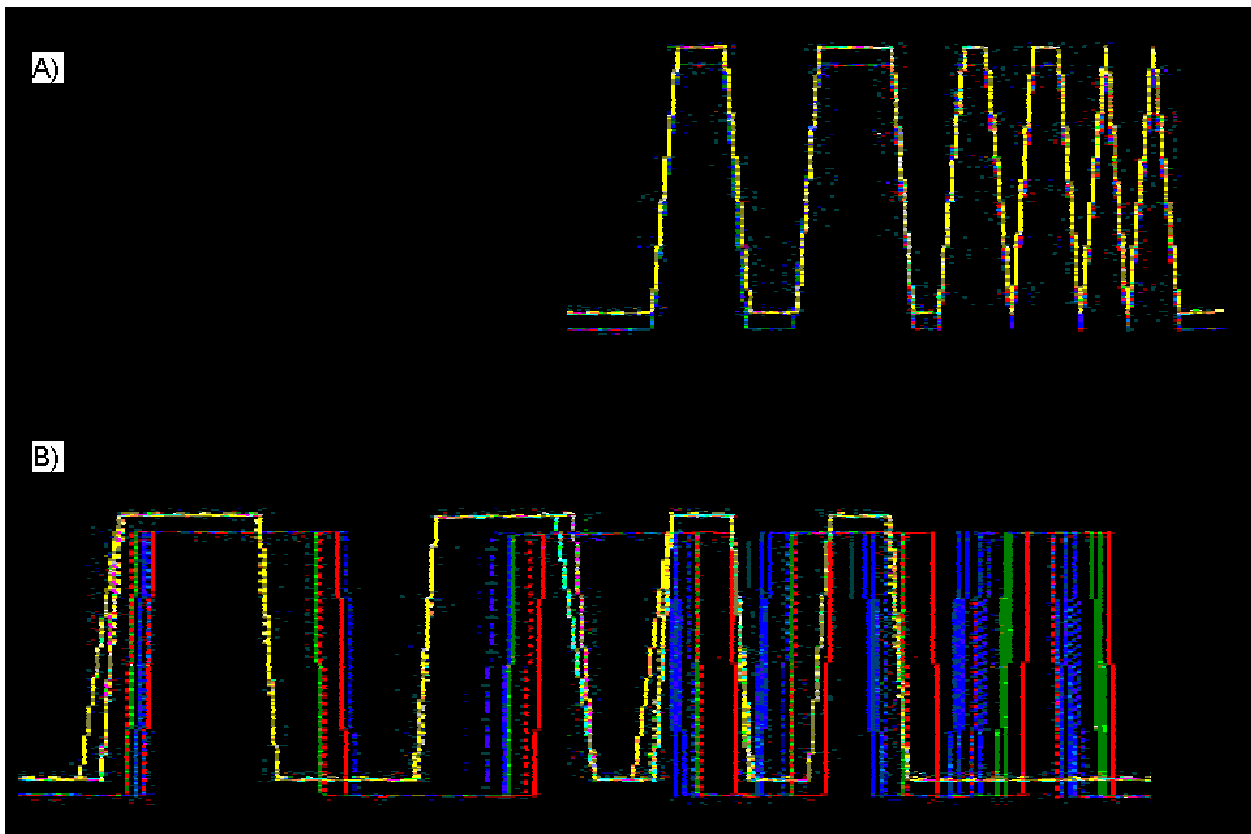
## 8.2. The strange jumps during (some of) ELMBs calibration



### 8.3. PVSS application speed test

#### A) Test of Direct Mode speed

~350 Voltages are switching every: 80 sec, 40 sec, 20 sec.



#### B) Test of Network Mode speed (OPC server)

~350 Voltages are switching every: 160 sec, 80 sec, 40 sec.

#### 8.4. The archive data in PVSS OPC-server mode (for 40 sec)

Name: SCT.Layer01.Loop01.Stave01.CoolTemp02,  
Addr: SCTdcsCOOL11:h0\_n0\_p0\_Unit01.Chan01

2002.02.12 21:48:45.830,	-5.85718		
2002.02.12 21:48:47.931,	-5.87918		
2002.02.12 21:48:53.659,	-5.89017		
2002.02.12 21:48:55.407,	-5.85718		
2002.02.12 21:49:01.457,	-5.87368		
2002.02.12 21:49:03.439,	-5.89017		
2002.02.12 21:49:09.581,	-5.88468		
2002.02.12 21:49:12.360,	-5.87368		
2002.02.12 21:49:17.458,	-5.87918		
2002.02.12 21:49:19.333,	-5.88468		
		2002.02.12 21:49:24.902,	14.9893
2002.02.12 21:49:27.251,	-5.87918		
		2002.02.12 21:49:33.323,	14.9728
		2002.02.12 21:49:35.508,	14.9893
		2002.02.12 21:49:41.157,	15.0004
		2002.02.12 21:49:43.507,	14.9728
		2002.02.12 21:49:49.390,	14.9838
		2002.02.12 21:49:52.091,	15.0004
		2002.02.12 21:49:57.151,	14.9893
		2002.02.12 21:49:59.622,	14.9838
		2002.02.12 21:50:04.655,	14.9949
		2002.02.12 21:50:07.525,	14.9893
		2002.02.12 21:50:13.688,	14.9783
		2002.02.12 21:50:15.978,	14.9949
		2002.02.12 21:50:21.193,	14.9949
		2002.02.12 21:50:23.621,	14.9783
		2002.02.12 21:50:29.200,	15.0225
		2002.02.12 21:50:32.603,	14.9949
		2002.02.12 21:50:37.826,	15.0004
		2002.02.12 21:50:40.107,	15.0225
		2002.02.12 21:50:44.857,	15.017
		2002.02.12 21:50:47.651,	15.0004
		2002.02.12 21:50:53.505,	15.0114
		2002.02.12 21:50:56.073,	15.017
		2002.02.12 21:51:01.521,	15.017
		2002.02.12 21:51:04.329,	15.0114
2002.02.12 21:51:10.174,	-5.87368		
		2002.02.12 21:51:13.341,	15.017
2002.02.12 21:51:18.571,	-5.89567		
2002.02.12 21:51:21.091,	-5.87368		
2002.02.12 21:51:25.722,	-5.87918		